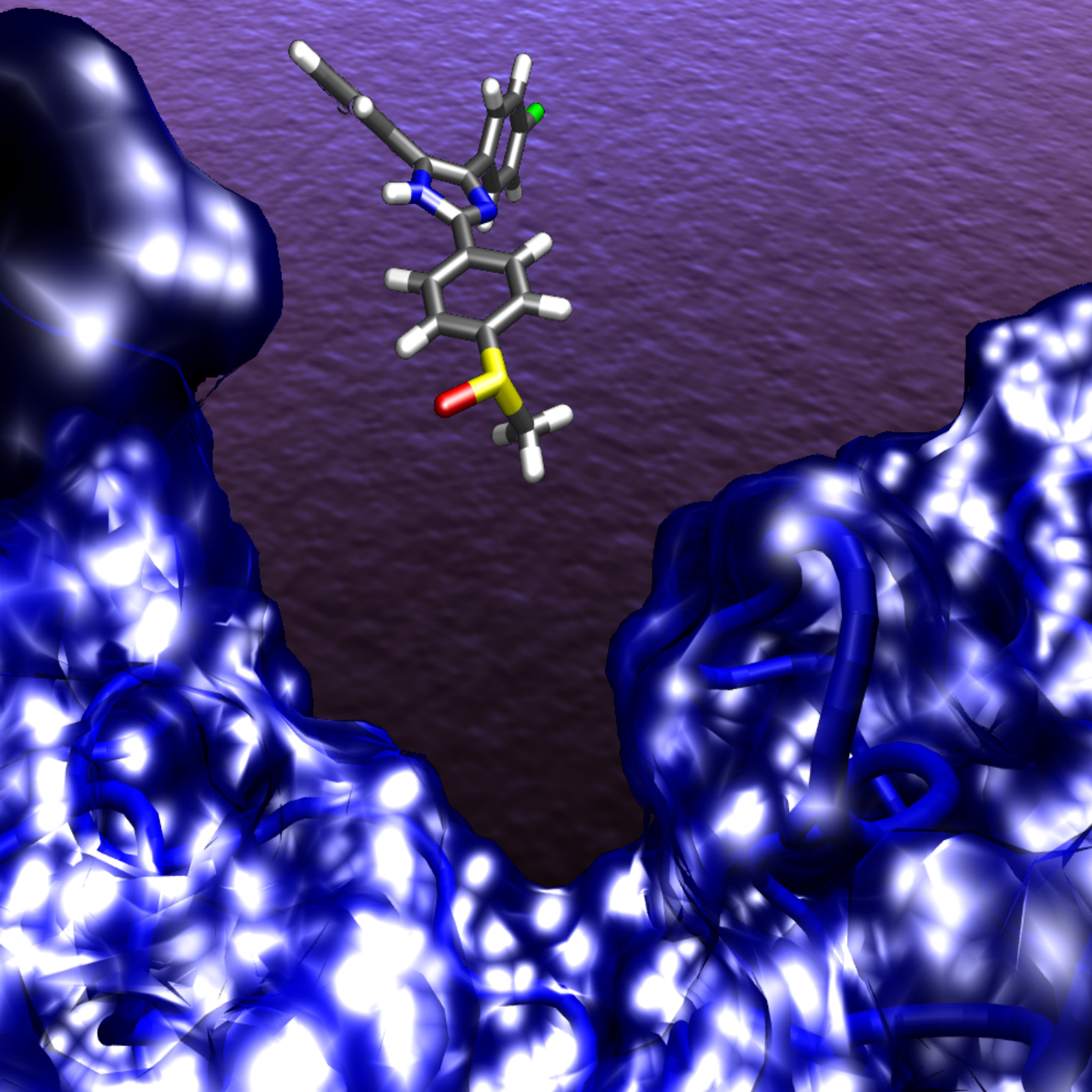
ProtoMS 2.22

User Manual



Acknowledgements

ProtoMS could not have been produced without the help and support of a large number of people. Special thanks go to Jon Essex and Mike King, who have both provided extensive support and understanding (especially as the appearance of ProtoMS was a bit of a surprise to them...).

Extra thanks go to those people who have helped me program ProtoMS. These include Julien Michel, who enjoys stressing ProtoMS past breaking point, and who is implementing many features that we hope get into the next release. Thanks to Sebastien and Caterina who are using ProtoMS to develope new science, and thanks again to Mike King who has helped with the debugging!

Thanks to Celltech R&D and the University of Southampton for their wise decision to allow me to release this code under the GPL. It was knowing that I could release this code freely that provided me with the motivation to put in all the extra time needed to get it ready for use. Speaking of the GPL, thanks to all of the free software projects that provided the software that I used to prepare ProtoMS and this manual. These include, but are not limited to, Linux, the Free Software Foundation (for G77, GCC, GLIBC amongst others), LaTeX, and pdfTeX, the Gimp, Python, Perl and Povray. The artwork has been produced using VMD (Humphrey, W., Dalke, A. and Schulten, K., 'VMD - Visual Molecular Dynamics', J. Molec. Graphics 1996, 14.1, 33-38).

Finally, thanks to the BBSRC and Celltech R&D who provided the funding to my PhD, during which the bulk of the original code was written, and again to Celltech R&D who provided extra funding to update the code from version 1.0 to 2.0 and write this manual.

The cover picture shows the SB1 inhibitor about to bind into the active site of p38 kinase. ProtoMS was originally written to simulate this system, and so you could say that p38 provided the inspiration for ProtoMS. The results of these simulations are presented in my PhD thesis.

License

ProtoMS is not a professional molecular modelling package and comes with NO WARRANTY, nor with any guarantee of accuracy or reliability. ProtoMS has been designed to calculate the properties of biomolecules, but there is no guarantee that it will calculate them correctly, and no support is available to help you resolve any errors. If you require support, then I recommend that you use a professional molecular modelling package, e.g. MCPRO from http://zarbi.chem.yale.edu/products/-mcpro/index.shtml

ProtoMS is distributed under the GNU General Public License (GPL). A copy of this license can be found in the 'COPYING' file, or in the appendix of this manual. The aim of this license is to allow anybody to freely copy and share the ideas contained within ProtoMS while at the same time preventing this program from being locked away within a piece of commercial software. The GPL achieves this aim as it grants you the right to copy, modify and distribute this code, *as long* as you do not try to limit the rights of others to do the same. This means that if you implement a new feature within ProtoMS, and then give a copy of the modified code to a third party, you cannot prevent the third party from giving the modified code to whomever they choose. If a patent, intellectual property, non-disclosure or employment contract prevents the third party from distributing your modified code, then they would not have been permitted to receive your modified code. Please note that the GPL does not state the you must give your code away to whomever asks! It merely states that whomever receives your code must be freely allowed to give it to whomever they choose. Also note that the GPL applies to both binary and source distribution, with the distinction that a third party who receives a binary copy of your code is entitled to ask you for the original source code, subject to a reproduction and distribution charge. The GPL is very specific in stating that the original source code is the form of the code that the developer uses to write the software. It is not acceptable to distribute name-mangled or otherwise obfuscated source code.

The choice of the GPL may lead to some legal obstacles to the use of ProtoMS for research collaborations between industrial and academic partners. These obstacles should however be easy to overcome if you remember that the GPL does not force you to give your code away, but rather that it prevents you from stopping recipients of your code from giving it away. While I am not a lawyer, and what follows is merely my opinion, I believe that you can safely use ProtoMS in the following scenarios;

1. If you are a user of ProtoMS then you are free to use ProtoMS in any way you desire. The GPL only applies to the software, not to your input or output files or research. If you publish any work that was produced with ProtoMS then I would appreciate a reference ("ProtoMS 2.0. C. J. Woods 2004"), but you are under no obligation to do so.

2. If you are a student or researcher developing new science in ProtoMS then again you are free to use ProtoMS in any way that you desire. If you modify the source code of ProtoMS then you may not give your modifications away without first checking that your University (and industrial collaborators) are happy for your modifications to be released under

the terms of the GPL. If they are not happy, then you would need to reimplement your ideas in another program before you could distribute them, though you could use your implementation in ProtoMS as a benchmark. You would then be prevented from distributing your original implementation in ProtoMS.

If you publish work based on your modified version of ProtoMS then if you refer to ProtoMS in your publication, then you must make it clear that you used a customised version of ProtoMS. The publication of work based on ProtoMS does not place you under any obligation to distribute your modifications. However, if your publication describes the new algorithm that you developed, then you may not prevent a third party from implementing that algorithm within ProtoMS. The only exception to this is if you have applied for a patent for the new algorithm, though this would have to have been made clear within your publication. If this is the case, then I would appreciate it if you would grant a royalty free license to ProtoMS and its derivatives, though you under no obligation to do so.

3. If you are the supervisor of a research group then you are free to allow the members of your group to use ProtoMS in any way that you desire. In addition, for the purpose of this license, your entire research group is viewed as a single party. Thus modifications to ProtoMS made by your group may be shared amongst members of the group without the need for these modifications to be placed under the GPL. These modifications would only need to be placed under the GPL if they were distributed to persons outside your research group. Note that any industrial collaborators that you work with are not technically part of your research group, and thus distribution of any modifications to your industrial collaborators would require that these modifications are placed under the GPL. I am however willing to overlook this and allow distribution of your modifications to your industrial collaborators on the condition that this distribution is purely for the purposes of testing and evaluation. If your industrial collaborators plan to exploit your modifications commercially (even within their own company), then they must either agree to release the modifications under the terms of the GPL, or reimplement your ideas in another software package, in which case you would be prevented from distributing your modifications to third parties.

4. If you are an industrial researcher collaborating with an academic research group who are using ProtoMS, then you can allow the academic research group to develop new science in ProtoMS, though you must make it clear that any modification to ProtoMS must not be distributed to third parties. I am willing to allow the academic researchers to distribute their modified version of ProtoMS to you for the purposes of evaluation and testing. This means that you may use the modified version of ProtoMS for in-house evaluation of the new science. If your company decides that it wishes to commercially exploit the new science, either by using it in drug discovery or by selling or patenting it, then your company must either agree to release the modification under the GPL, and grant a royalty free patent for the new idea to ProtoMS and any derived version of ProtoMS, or you must reimplement the new science in another software package. If you decide to reimplement, then you are prevented from distributing or using the modified version of ProtoMS and you must inform the academic research group that they are also prevented from distributed the modified version of

ProtoMS.

ProtoMS is short for “Prototype Molecular Simulation”. As long as you view ProtoMS as a piece of software for rapid prototyping of new science, and not for commercial exploitation of new science, then the GPL license should give you very few problems.

Finally, one of the conditions that I agreed to when I obtained permission to distribute this code was that I would not commercially exploit it. This means that even if I wanted to sell you this code, I couldn’t, so please don’t ask.

Contents

Chapter 1

Introduction

ProtoMS is short for “Prototype Molecular Simulation”, and is a software package that was originally designed by Christopher Woods to perform protein-ligand binding free energy calculations during his PhD. Julien Michel latter added numerous features and used the program extensively during his PhD. The program is now routinely used by several members of the research group of Jonathan Essex. This document has been written to try and explain how to use ProtoMS.

This document is presented as a hyperlinked PDF file, and this is the format that I recommend you use when you read it. This is because this document may change between versions of ProtoMS, and you do not want to have to keep printing it out, and viewing it electronically means that you can take advantage of the search facilities of your PDF viewer, and you can click on highlighted links to jump around between sections. You should also see bookmarks for all of the chapters, sections and subsections in the left pane, and may also see thumbnails of all of the pages.

The user manual has been written as a reference manual, with extensive hyperlinking to allow you to quickly dip in and out to find the information you need. While you could read it from start to end, it would be a boring and repetitive read and you probably wouldn’t learn much! I recommend that you play with the examples that come with ProtoMS. You can then use the links in those descriptions to dip in and out of the user manual, thus obtaining a more detailed knowledge of how the examples, and thus ProtoMS, work.

1.0.1 References

If you want to read about studies that have been conducted with ProtoMS in its various versions you can consult the following scientific papers:

1. Efficient generalized Born models for Monte Carlo simulations. Michel, J. ; Taylor, R. D. ; Essex, J. W. J.

Chem. Theory Comput., 2 (3), 732 -739, 2006. This paper describes the implementation of the Generalized Born Surface Area force field in ProtoMS 2.2 and the various techniques that can be used to increase efficiency.

2. Protein-ligand binding affinity predictions by implicit solvent simulations: a tool for lead optimization? Michel, J. ; Verdonk, M. L. ; Essex, J. W. J. Med. Chem. , 49 (25), 7427 -7439, 2006. This paper discusses the use of implicit solvation techniques in the computation of binding affinity and compare the predictions to explicit solvent simulations, for a set of 38 inhibitors and 3 proteins.

3. Classification of Water Molecules in Protein Binding Sites C. Barillari, J. Taylor, R. Viner, J.W. Essex J. Am. Chem. Soc., 129(9), 2577-2587, 2007. This paper demonstrates the use of free energy techniques to predict whether or not water molecules present in protein binding site are tightly or loosely bound.

4. Protein-ligand complexes: computation of the relative free energy of different scaffolds and binding modes. Michel, J. ; Verdonk, M. L. ; Essex, J. W. J. Chem. Theory Comput., in press. This paper describes the dual topology technique available in ProtoMS 2.2 that allows to attempt the computation of relative free energies between molecules that differs significantly in topology.

1.0.2 Formatting

The following formats are used throughout this document. Program commands or contents of files will be written in shaded boxes, e.g.

```
temperature float
```

where *float* is a floating point option to the command. If this option is given a value (e.g. 25.0, then it is written like this;

```
temperature 25.0
```

The following options are standard to many commands;

float A floating point number.

integer An integer. Most integer options given to ProtoMS are positive integers, greater than 0. This will always be made clear with the command.

logical A logical, true or false option. Possible values for this option are `true` or `false`, `yes` or `no` or `on` or `off`, depending on your personal preference.

filename This is the name of a file. Note that while ProtoMS is mostly case insensitive, file handling is dependent on the operating system you are using, so the filenames may be case sensitive. UNIX/Linux are examples of operating systems where case is important, while case is not important for Windows.

Chapter 2

Compilation and Installation

The ProtoMS package supplies the following files and directories;

`README` File that contains brief installation instructions for ProtoMS, and any last minute addendums or errata that arrived too late to make it into the manual!

`tutorials` This directory contains a number of examples that demonstrate applications of ProtoMS.

`interfaces` This directory contains interfaces for ProtoMS to common scripting languages. Currently interfaces to the Perl and Python languages are provided.

`manual.pdf` This manual!

`parameter` This directory contains all of the standard parameter files that describe the standard forcefields implemented in ProtoMS.

`src` This directory contains all of the source code.

2.1 Programming Language

ProtoMS is written in slightly extended Fortran 77. The extensions used are;

The maximum line length is up to 132 characters, rather than 72.

Variable, subroutine and function names are greater than 6 characters.

`do/enddo` loops are used rather than `do/continue`.

Fortran `include` is used to include the contents of other files.

The `flush`, `getarg` and `getenv` non-standard intrinsic functions are used.

ProtoMS performs string manipulation using the `len` function. In addition, the string manipulation assumes the same string handling behaviour as the GNU Fortran compiler (`g77`), so there is the possibility of strange formatting bugs when using different compilers.

The `Date_And_Time` Fortran 90 intrinsic subroutine is used to get the current time. This is used to provide a default seed to the random number generator. This can be removed by commenting out the relavant lines in `getoptions.F`, though you will need to provide a random number seed manually.

ProtoMS has been written using the GNU Fortran compiler, `g77`, version 3.3.4, on the Linux operating system. ProtoMS is thus known to work well with g77 and Linux. ProtoMS has also been compiled and tested using the Portland Group Fortran Compiler. ProtoMS has been compiled with other compilers but not extensively tested. It is therefore advised to use g77 of pgf77 with ProtoMS.

Portland Group Fortran Compiler

Portland Group Fortran Compiler works well with ProtoMS, with the only problem being that I could not find out how to make it support the Date_and_Time Fortran 90 intrinsic when compiling Fortran 77 code. An alternative subroutine getdateandtime.F.pgf77 is provided. You should copy this file to getdateandtime.F before compiling with pgf77.

2.2 Building ProtoMS

Building ProtoMS should be straightforward if you have a Fortran compiler that supports the extensions described in section 2.1, and a version of `make` that supports the GNU Makefile format. Simply go into the `src` directory and edit the `Makefile` that you find there. This file contains a lot of comments to help you edit the file, and all you should need to do is edit the compilation flags to best optimise ProtoMS to your system. Once you have edited the `Makefile` you can then run `make`. After about 5 minutes, the compilation should hopefully finish, and the ProtoMS executable placed in the top directory. The executable will be called `protoms2` on UNIX/Linux, and `protoms2.exe` on Windows. This executable should be run from the command line, or via a script. You should then change into the `example` directory and try out some of the examples. You should also try some of the tests as well to ensure that your version of ProtoMS is working correctly.

Chapter 3

User Manual

ProtoMS is a powerful simulation program that is capable of being used in many different ways. ProtoMS was originally designed to perform Monte Carlo free energy calculations on protein-ligand systems, so a lot of the terminology and ideas associated with ProtoMS derive from protein-ligand Monte Carlo methodology. While the code was originally designed with this use in mind, the framework is sufficiently flexible to allow the study of a wide range of different systems, using a wide range of simulation methodology.

3.1 Design of ProtoMS

At the core of ProtoMS are four central concepts;

Proteins/Solutes/Solvents ProtoMS divides all molecules to be simulated into 'proteins', 'solutes' and 'solvent'.

Classical Forcefields ProtoMS Uses a generic classical forcefield to calculate the energy of the molecules. This forcefield may be specialised such that ProtoMS is able to implement a wide range of modern molecular mechanics forcefields.

Perturbation and λ ProtoMS provides support for free energy calculations by allowing forcefields and geometries to be perturbed using a λ coordinate. The forcefield for any protein, solute or solvent may be perturbed, and the geometry of any solute may be perturbed.

Generic Moves ProtoMS is designed around the concept a 'move'. The move can do anything, from a Monte Carlo translation of solvent to a docking type move on a solute. A simulation is constructed by stringing a collection of moves together.

3.1.1 Proteins / Solutes / Solvents

ProtoMS divides all of the molecules loaded within a system into solvents, solutes and proteins;

solvents A solvent is any rigid molecule. Solvents may only be translated and rotated, and by default, 10000 solvent molecules may be loaded, each consisting of up to 10 atoms. Solvent molecules do not have to be small - a rigid lipid molecule could be modelled as a solvent. There is no requirement for the solvents loaded in a system to be the same. Indeed every solvent loaded could be a different type of molecule!

solutes A solute is any flexible molecule. Solutes can be translated and rotated, and change their internal geometry. By default 25 solutes, each composed of 25 residues, each composed of 50 atoms may be loaded simultaneously. Solute molecules are described using z-matrices, thus a solute molecule is perhaps what you would be most familiar with from other Monte Carlo simulation programs. Note that you can describe a protein molecule as a solute, and that you do not need to load it up as a 'protein'.

proteins A protein is any flexible chain molecule (polymer). A protein is composed of a linear chain of residues, with interresidue bonds connecting one residue to the next. By default, ProtoMS can load up to 3 proteins simultaneously, each protein consisting of 500 residues, each consisting of up to 34 atoms.

Solvents

Solvents are loaded into ProtoMS from PDB files (see section 3.8.5). Each solvent molecule is identified by its residue name (the fourth column in the PDB file), e.g. ProtoMS identifies the TIP4P solvent with the residue name 'T4P'. ProtoMS loads the coordinates of the solvent from the PDB file, and then assigns the parameters for the solvent from a solvent template (see section 3.6.1). The solvent template contains the information necessary to identify all of the atoms in the solvent molecule and to assign forcefield parameters to each atom. Note that this version of ProtoMS uses the coordinates of the solvent molecule that are present in the PDB file. ProtoMS does not yet have the capability to modify these coordinates to ensure that the internal geometry of the solvent is correct for the solvent model. This means that as solvents are only translated and rotated, the internal geometry of the solvent molecule loaded at the start of the simulation will be identical to that at the end of the simulation.

ProtoMS can also generate a large solvent box by replicating a smaller solvent box. An equilibrated box of TIP4P molecules is provided with the program and a tutorial demonstrates how to use such feature.

Solutes

Solutes are also loaded into ProtoMS from PDB files (see section 3.8.4). Each solute molecule is identified by its solute name, which is given in the 'HEADER' record of the PDB file. ProtoMS obtains the coordinates of the solute from the PDB file, and will then find a solute template that matches this solute name (see section 3.8.2). The solute template is used to build the z-matrix for the solute, and to assign all of the forcefield parameters. The solute template is also used to assign the connectivity of the solute and to define the flexible internal coordinates. The solute molecule is constructed using the z-matrix, with the reference being three automatically added dummy atoms, called 'DM1', 'DM2' and 'DM3', all part of residue 'DUM'. These dummy atoms are automatically added by ProtoMS at the geometric center of the solute, as a right angled set of atoms pointing along the major and minor axes of the solute.

Proteins

Proteins are loaded into ProtoMS via PDB files (see section 3.8.3). Each PDB file may only contain a single protein chain. ProtoMS constructs the linear chain of molecules based on the order of residues that it reads from the PDB file, and will ignore the residue number read from the PDB file. This means that you must ensure that you have the residues ordered correctly within the PDB file. ProtoMS assigns to each residue both a chain template (see section 3.8.2), that describes the backbone of the residue, and a residue template (see section 3.8.2), that describes the sidechain. The residue template is located based on the name of the residue given in the fourth column in the PDB file (e.g. 'ASP' or 'HIS'). The chain template is located based on the chain template associated with the residue template for the position of the residue within the chain. For example, residue 'ASP' has a standard amino acid backbone chain template if this residue was in the middle of the chain, an NH_3^+ capped backbone chain template if this was the first residue of the chain (and thus at the n-terminus), and a CO_2^- capped backbone chain template if this were the last residue of the chain (and thus at the c-terminus). If the protein consisted of only one residue, then the zwitterionic amino acid chain template would be used for 'ASP'.

ProtoMS obtains the coordinates of each residue from the PDB file, and will then use the residue and chain templates to build the z-matrix for each residue, and to assign all of the forcefield parameters.

Proteins are moved in a different manner in ProtoMS compared to other Monte Carlo packages that are available. Each residue is moved independently, using both the internal geometry moves defined by the template z-matrix, and by backbone translation and rotation moves of the chain atoms (see figure 3.1).

Four special backbone atoms (bbatoms) are identified in the chain-backbone of each residue. These atoms form the reference from which the rest of the residue atoms are built. These four atoms can be translated and

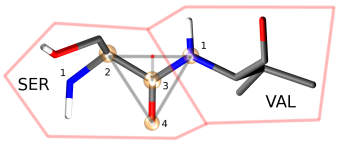


Figure 3.1: Four atoms from each protein residue are designated as backbone atoms (`bbatoms`). For most residues these atoms are the N, CA, C and O atoms respectively. The four backbone atoms for two neighbouring residues are shown above. The protein backbone move moves the last three bbatoms of one residue and the first bbatom of the next residue. This is because the moves assumes that these four bbatoms form a rigid triangle (as is shown by the grey lines). The four atoms are translated and rotated as a rigid triangle, with the origin of rotation of the triangle centered on the intersection of the vector between bbatoms 2 and 1, and the vector between bbatoms 3 and 4 (marked as a red dot directly above the C=O bond). Because this triangle is translated and rotated as a rigid unit, all atoms connected to the atoms of this triangle will also be translated and rotated as a rigid unit.

Parameter	Description	Value
MAXPROTEINS	Maximum number of proteins	3
MAXRESIDUES	Maximum number of residues per protein	500
MAXSCATOMS	Maximum number of atoms per protein residue	30
MAXSOLUTES	Maximum number of solutes	25
MAXSOLUTERESIDUES	Maximum number of residues per solute	25
MAXSOLUTEATOMSPERRESIDUE	Maximum number of solute atoms per residue	50
MAXSOLVENTS	Maximum number of solvent molecules	10000
MAXSOLVENTATOMS	Maximum number of atoms per solvent	10

Table 3.2: The default value of the maximum number of proteins, solutes and solvents that may be loaded simultaneously by ProtoMS. These values may be changed by editing the `dimensions.inc` file located in the `src` directory, and recompiling ProtoMS.

rotated as a rigid unit via protein backbone moves (see figure 3.1). As the rest of the residue is constructed from these bbatoms, the rest of the residue is thus also translated and rotated. Because the bbatoms are translated and rotated as a rigid unit, the internal geometry of these backbone atoms are held constant throughout the simulation. This means that the internal geometry of the bbatoms is taken from the PDB file, and may not be modified by the chain or residue templates. It is also not possible to build missing bbatoms, so they must all be present in the PDB file.

Once the coordinates and z-matrices of each residue have been assigned, interresidue bonds are added between the first bbatom of each residue and the third bbatom of the previous residue (e.g. for 'ASP', bonds would be added from the 'N' atom of the 'ASP' residue to the 'C' atom of the preceeding amino acid residue). If the length of this bond is less than 4 Å then this bond is added as a real bond, and its energy is evaluated as part of the forcefield. However, if the length is greater than 4 Å, then this bond will be added as a dummy bond, and a warning message output. This is useful in cases where you wish to load up a protein scoop, e.g. from around the active site. This option should be used with care in conjunction with backbone moves.

Limits

ProtoMS is written using slightly extended Fortran 77 (see chapter **??**). This means that the maximum numbers of loaded proteins, solutes and solvents has to be set at compile time.

Table 3.2 gives the default values for the maximum number of proteins, solutes and solvents. Please note that you may change these numbers to fit the system that you are interested in, e.g. if you were investigating a single protein in a lipid bilayer then you may choose to model the lipid as a solute (thus requiring a large increase in the number of solute molecules, but a decrease in the number of solute residues), and you could reduce the maximum number of protein molecules to one. By balancing the numbers of protein, solutes and solvents you should find

that you are able to load up the system that you want to simulate.

3.1.2 Classical Forcefields

ProtoMS was designed to perform simulations using a range of different molecular mechanics (MM) forcefields. To achieve this aim, a generic forcefield has been implemented, and this can be specialised into a specific, traditional forcefield.

The forcefield in ProtoMS is comprised of several terms;

Intermolecular Potential

An intermolecular potential acts between all molecules within the system. The intermolecular potential between a pair of molecules, A and B, $U_{molecule}(A,B)$, with A consisting of n_A atoms and B consisting of n_B atoms, is formed as the sum of the non-bonded potential, $U_{nb}(i,j)$ between each pair of atom sites, i and j, between the two molecules, scaled by a constant, scl, e.g.

$$U_{molecule}(A,B) = scl(R) \times \left(\sum_{i=1}^{n_A} \sum_{j=1}^{n_B} U_{nb}(i,j) \right), \tag{3.1}$$

where R is the shortest distance between a pair of atom sites between the molecules. The scaling factor is set according to

$$\begin{aligned} &R \geq r_{cut} \rightarrow scl = 0.0 \\ &r_{cut} - r_{feather} \leq R \leq r_{cut} \rightarrow scl = \frac{r_{cut}^2 - R^2}{r_{cut}^2 - (r_{cut} - r_{feather})^2} \\ &R \leq r_{feather} \rightarrow scl = 1.0, \end{aligned} \tag{3.2}$$

where r_{cut} and $r_{feather}$ are the non-bonded cutoff and feather parameters.

The non-bonded potential between the pair of atoms is evaluated as the sum of the Coulombic and Lennard-Jones (LJ) potentials between the atoms,

$$U_{nb}(i,j) = \frac{q_i q_j}{4\pi\varepsilon_0 r(i,j)} + 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r(i,j)} \right)^{12} - \left(\frac{\sigma_{ij}}{r(i,j)} \right)^{6} \right], \tag{3.3}$$

where q_i and q_j are the partial charges on the two atom sites, $r(i,j)$ is the distance between the atom sites, ε_0 is the permittivity of free space and σ_{ij} and ε_{ij} are the Lennard Jones parameters for the atom site pair i and j.

The LJ parameters for an atom site pair are calculated as the average of the LJ parameters for the same site pair, e.g.

$$\sigma_{ij} = 0.5 \times (\sigma_{ii} + \sigma_{jj}). \tag{3.4}$$

Either the arithmetic average (shown in equation 3.4) is used, or the geometric average is used, e.g.

$$\varepsilon_{ij} = \sqrt{\varepsilon_{ii} \times \varepsilon_{jj}}. \tag{3.5}$$

The AMBER family of forcefields use the arithmetic average for σ, and the geometric average for ε, while the OPLS family of forcefields use the geometric average for both parameters.

The intermolecular potential is formed as the sum of the non-bonded potential over all pairs of atom sites. It should be noted that an atom site does not necessarily need to lie at the center of each atom, and it may lie between atoms, or at the location of any lone pairs. Individual atoms may possess many atom sites, or even no atom sites.

Bond Potential

A bond potential acts over all of the explicitly added, non-dummy bonds within a molecule. ProtoMS makes no attempt to find any implicit bonds within a molecule, and it is not possible to add a bond between atoms of different molecules. The energy of each bond, U_{bond}, is evaluated according to

$$U_{bond}(r) = k_{bond}\left(r - r_0\right)^2, \tag{3.6}$$

where r is the bond length, k_{bond} is the force constant for the bond, and r_0 is the equilibrium bond length. The total bond energy of a molecule is the sum of the bond energies for all of the bonds within the molecule, and the total bond energy of the system is the sum of the bond energies for each of the molecules in the system.

Angle Potential

An angle potential acts over all angles between atoms that are connected by non-dummy bonds, and over all non-dummy angles that have been explicitly added to the molecule. The energy of each angle, U_{angle}, is evaluated according to

$$U_{angle}(\theta) = k_{angle}\left(\theta - \theta_0\right)^2, \tag{3.7}$$

where θ is the size of the angle, k_{angle} is the force constant for the angle, and θ_0 is the equilibrium angle size. The total angle energy of a molecule is the sum of the angle energies for each of the angles within the molecule, and

the total energy of the system is the sum of the angle energies for each of the molecules in the system.

Urey-Bradley Potential

A Urey-Bradley potential may act between the first and third atoms of some of the angles that are evaluated for the angle potential. If this is the case, then a Urey-Bradley energy is added onto the angle energy. The Urey-Bradley energy, U_{uby}, is evaluated according to

$$U_{uby}(x) = k_{uby}\big(x - x_0\big)^2, \tag{3.8}$$

where x is the distance between the first and third atoms, k_{uby} is the Urey-Bradley force constant, and x_0 is the equilibrium distance.

Dihedral Potential

A dihedral potential acts over all dihedrals between atoms that are connected by non-dummy bonds, and over all non-dummy dihedrals that have been explicitly added to the molecule. Such explicitly added dihedrals may be used to add improper dihedrals that maintain the stereochemistry of chiral centers. The energy for each dihedral, $U_{dihedral}$, is formed as the sum of n cosine terms,

$$U_{dihedral}(\phi) = \sum_{i=1}^{n} k_{i1}\left[1.0 + k_{i2}\big(cos(k_{i3}\phi + k_{i4})\big)\right], \tag{3.9}$$

where k_{i1} to k_{i4} are dihedral parameters and ϕ is the size of the dihedral. The total dihedral energy of a molecule is the sum of the dihedral energies for each of the dihedrals in the molecule, and the total dihedral energy of the system is the sum of the dihedral energies of each of the molecules.

Intramolecular non-bonded Potential

An intramolecular non-bonded potential acts between all intramolecular pairs of atoms that are either not connected by a non-dummy bond, or are not both connected to a third atom by a non-dummy bond. To make this more clear, if two atoms are connected by a non-dummy bond then they are said to be 1-2 bonded. If two atoms are both connected to a third atom by non-dummy bonds, then they are said to 1-?-3, or 1-3 bonded. Similarly, if the pair of atoms are connected together via two atoms via non-dummy bonds, then they are said to be 1-?-?-4, or 1-4 bonded. An intramolecular non-bonded potential does not act over 1-2 or 1-3 bonded pairs within a molecule, but does act over 1-4 bonded pairs and above. Note that ProtoMS only looks at the non-dummy bonds between atoms,

and will not consider whether or not there are non-dummy angles, Urey-Bradley or dihedral terms involving these atoms.

The intramolecular non-bonded potential of a molecule, U_{intra} is the sum of the non-bonded energy between all 1-5 and above pairs of atoms within the molecule, plus the sum of the non-bonded energy between all 1-4 atoms scaled by a 1-4 scaling factor, e.g.

$$U_{intra} = \sum_{\text{1-5+ i j pairs}} U_{coul}(i,j) + U_{lj}(i,j) + \sum_{\text{1-4 i j pairs}} scl_{coul} U_{coul}(i,j) + scl_{lj} U_{lj}(i,j), \tag{3.10}$$

where,

$$U_{coul}(i,j) = \frac{q_i q_j}{4\pi\varepsilon_0 r}, \tag{3.11}$$

and

$$U_{lj}(i,j) = 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right]. \tag{3.12}$$

Equations 3.11 and 3.12 are the Coulomb and Lennard Jones equations, as seen in the intermolecular potential in equations 3.1 and 3.3. scl_{coul} and scl_{lj} are the Coulomb and Lennard Jones scaling factors.

Generalized Born Surface Area potential

While free energy simulations are usually conducted in explicit solvent, ProtoMS supports Generalized Born Surface Area (GBSA) implicit solvent models. Relatively few free energy implicit solvent studies have been conducted and such option should be tested carefully before embarking onto expensive free energy simulations. The GBSA theory assumes that the total solvation free energy of a molecule A is a sum of a polar and non-polar energy term:

$$\Delta G_{solv} = \Delta G_{pol} + \Delta G_{nonpol} \tag{3.13}$$

The second term, is simply proportional to the solvent accessible surface area (SASA) of the molecule, times a parameter that depends on the atom types present in the molecule. The first term is more complex and derived

from the following equation :

$$\Delta G_{pol} = -\frac{1}{2}\left(\frac{1}{\varepsilon_{vac}} - \frac{1}{\varepsilon_{solv}}\right)\sum_i\sum_j \frac{q_i q_j}{\sqrt{r_{ij}^2 + B_i B_j e^{\frac{-r_{ij}^2}{4B_i B_j}}}} \tag{3.14}$$

ε_{vac} and ε_{solv} are the dielectric constants of the vacuum and the solvent respectively, q_i the atomic partial charge of atom i, r_{ij} the distance between a pair of atoms ij, and B_i is the effective Born radius of atom i.

The effective Born Radius B_i is in essence the spherically averaged distance of the solute atom to the solvent. An accurate estimate of this quantity is essential to calculate high quality solvation free energies. It is however fairly complex to compute as it formalyl involves an integral over the position of all the atoms in the system. While numerical techniques can calculate such value, they are too slow to be of practical use in a simulation. In ProtoMS, the effective Born radii are calculated using the Pairwise Descreening Approximation (PDA) method.

$$\frac{1}{B_i} = \frac{1}{\alpha_i} - \frac{1}{2}\sum_{j\neq i}\left[\frac{1}{L_{ij}} - \frac{1}{U_{ij}} + \frac{r_{ij}}{4}\left(\frac{1}{U_{ij}^2} - \frac{1}{L_{ij}^2}\right) + \frac{1}{2r_{ij}} ln\frac{L_{ij}}{U_{ij}} + \frac{S_j^2\alpha_j^2}{4r_{ij}}\left(\frac{1}{L_{ij}^2} - \frac{1}{U_{ij}^2}\right)\right] \tag{3.15}$$

$$L_{ij} = 1 \quad if \quad r_{ij} + S_j\alpha_j \leq \alpha_i$$

$$L_{ij} = \alpha_i \quad if \quad r_{ij} - S_j\alpha_j \leq \alpha_i < r_{ij} + S_j\alpha_j$$

$$L_{ij} = r_{ij} - S_j\alpha_j \quad if \quad \alpha_i \leq r_{ij} - S_j\alpha_j$$

$$U_{ij} = 1 \quad if \quad r_{ij} + S_j\alpha_j \leq \alpha_j$$

$$U_{ij} = r_{ij} + S_j\alpha_j \quad if \quad \alpha i < r_{ij} + S_j\alpha_j$$

In equation 3.15 r_{ij} is the distance between a pair of atoms ij and α_i is the intrinsic Born radius of atom i, that is, the Born radius that atom i would adopt if it was completely isolated. Finally S_j is a scaling factor which compensates for systematic errors introduced by this approximate Born radii calculation.

As the name says, the technique approximate the descreening (the extent to which a nearby atom j displaces a volume that would have otherwise been occupied by solvent) by a fast summation of pairwise terms. It is however not rigorous and has to be parameterised carefully to yield robust performance. The PDA method tend to systematically underestimate the Born radius of buried atoms because it incorrectly assign high dielectric constants to numerous small voids and crevices that exist between atoms in a protein and are not occupied by

water. To increase accuracy, a re-scaling technique has been implemented.

$$\frac{1}{B_i} = \frac{1}{\alpha_i} - Itanh\left(\alpha\psi - \beta\psi^2 + \gamma\psi^3\right) \tag{3.16}$$

where I is the summation term from the PDA calculation, ψ, α, β and γ are parameters taken from the litterature.

The rescaling option has not been used extensively in ProtoMS and should be used with caution. It appears it may prove useful when simulation buried protein binding sites.

The GBSA force field implemented in ProtoMS was parameterised to be used with the AMBER99 and the GAFF force fields. While alternative force fields could be used, a loss of accuracy could be expected.

GBSA simulations are order of magnitude more efficient than explicit solvent simulations of small isolated molecules. However, they slow down rapidly when the size of the system increases. This is especially notable in Monte Carlo simulations where a small movement of part of a system formally warrants the computation the entire solvation energy of the system. This issue arises because the GBSA energy terms are not strictly pairwise decomposable. It is possible to use however different techniques to increase the speed of a GBSA simulation. Cutoffs in the calculation of the Born radii are introduced and in addition the update of pairwise GB energies can be skipped if the Born radii of either atoms have not changed more than a certain threshold value after a MC move. Because this option will introduce energy drifts, it is advised to periodically recalculate rigorously the GB energy. In addition, a more complex Monte Carlo move is implemented in ProtoMS. This option allows to conduct a simulation with a crude GBSA model and a low cutoff for the non bonded energy terms. Normally the predicted macroscopic properties would suffer from such crude treatment of intermolecular energies. However, periodically, a special acceptance test is employed to remove the bias introduced by the crude potential and ensure that the equilibrium density of states generated by the Monte Carlo simulation converges to the equilibrium density of states suitable for the standard biomolecular potential.

Actual speedups using either techniques are system dependent and optimisation of the different parameters can be a complex task. It is advised to use the default parameters described latter in the manual.

Caveats

ProtoMS implements this forcefield mostly as described. However there are a few shortcuts that are taken to improve the efficiency of the code. These shortcuts are based on the three-way split of the molecules of the system into solvents, solutes and proteins;

solvents As solvents are rigid, there is no need to evaluate any of the intramolecular potentials. ProtoMS thus only evaluates the intermolecular energy of solvent molecules.

solutes ProtoMS evaluates the forcefield of solute molecules exactly as described, with no shortcuts.

proteins ProtoMS implements a protein as a chain of residues. As these molecules can be large, and typically larger than the non-bonded cutoff, ProtoMS implements the non-bonded cutoff differently for proteins. Instead of evaluating the non-bonded cutoff for the protein as a whole, ProtoMS implements a residue-based cutoff, with the cutoff scaling factors evaluated individually for each residue. Additionally, the intramolecular non-bonded energy is also scaled according to the non-bonded cutoffs given in equation 3.3. If you do not want to use residue based cutoffs, then it is possible to tell ProtoMS to use a molecule based cutoff, in which case the forcefield for proteins will be evaluated exactly as described with no shortcuts.

3.1.3 Perturbation and λ

ProtoMS is capable of calculating the relative free energy of two systems. ProtoMS does this by perturbing one system into the other through the use of a λ-coordinate. If A and B are the two systems of interest, then the forcefield is constructed such that at $\lambda = 0.0$ the forcefield represents system A, at $\lambda = 1.0$ the forcefield represents system B, and at λ value inbetween, the forcefield represents a hybrid of A and B.

ProtoMS implements two methods of perturbing between systems A and B;

Single topology System A is perturbed into system B by scaling the forcefield parameters such that the model morphs from A to B.

Dual topology System A and B are simulated together, with λ scaling the total energies of A and B such that one system is turned off as the other is turned on.

Single Topology Calculations

ProtoMS assigns two sets of parameters to every single forcefield term; one parameter represents that term at $\lambda = 0.0$ (par_0), the other represents that term at $\lambda = 1.0$ (par_1). λ is used to linearly scale between these two parameters to obtain the value of the parameter at each value of λ (par_λ);

$$par_\lambda = (1.0 - \lambda) \times par_0 + \lambda \times par_1. \tag{3.17}$$

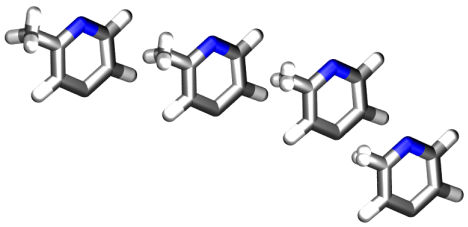


Figure 3.2: Geometry variations allow for a smoother transition between two systems, for example here a methyl group is smoothly converted into a hydrogen.

This equation is used to scale the charge, σ and ε parameters assigned to each atom site (see equations 3.11 and 3.12), and the force constants (k_{bond}, k_{angle} and k_{uby}) and equilibrium sizes (r_0, θ_0 and x_0) for the bond, angle and Urey-Bradley terms (see equations 3.6, 3.7 and 3.8). This equation is not used to scale the dihedral parameters, as the functional form of the dihedral potential is more complicated. Rather than scale the dihedral parameters, ProtoMS uses λ to scale the total energy of each dihedral;

$$U_{dihedral}(\phi)_\lambda = (1.0-\lambda) \times U_{dihedral}(\phi)_0 + \lambda \times U_{dihedral}(\phi)_1, \tag{3.18}$$

where $U_{dihedral}(\phi)_0$ is the dihedral energy using the parameters for $\lambda = 0.0$, $U_{dihedral}(\phi)_1$ is the dihedral energy using the parameters for $\lambda = 1.0$, and $U_{dihedral}(\phi)_\lambda$ is the scaled dihedral energy at that value of λ.

Any and all parts of the forcefield can be scaled. This includes all of the forcefield parameters of any solutes, all of the parameters of any proteins, and all parameters of any solvent molecules. While this is very useful, and enables perturbations of any and all parts of the system, there are many cases where just changing the forcefield parameters is not sufficient to smoothly morph from one system into the other. There are many cases where the geometry of the molecules needs to be changed with λ. Fortunately ProtoMS provides this capability for solute molecules. Any internal coordinates that are part of the z-matrix of a solute molecule may perturbed with λ. Geometry variations are a powerful tool as they allow for very complicated, yet very smooth transitions between two systems to be described. A good example of such a transition is the annihilation of the hydrogen atoms as a methyl group is morphed into a single hydrogen (see figure 3.2).

As well as enabling smooth transitions between systems, geometry variations may be used to calculate potentials of mean force along structural coordinates. An example of this use for calculating the potential of mean force along an intermolecular separation axis is given in section **??**. An example of using this feature to perform a torsion drive is also given in section **??**.

Dual Topology Calculations

A dual topology method to calculate free energy changes is also available in ProtoMS. In the single topology method force field terms were linearly interpolated so that they match the force field parameters suitable for particular molecule at either end of the perturbation (λ 0.0 or λ 1.0). As two molecules often differ not only in their force field terms but also their geometry, it is often necessary to modify the internal coordinates as well. This is relatively easy In simple cases (morphing a methyl group into a hydrogen group) but for larger, complex, perturbations this is often cumbersome if not impossible. In the dual topology method no geometry variations are attempted. However, the interaction energy of a pair of solutes with their surroundings (solvent, protein, other solutes), is gradually turned on or off with the coupling parameter.

$$U(\lambda) = U_0 + \lambda U(S_2) + (1-\lambda)U(S_1) \tag{3.19}$$

Equation 3.19 thus shows that at any given value of λ, the total energy of the system consists in a term U_0 that is independent of the perturbation and a term $U(S_2)$ and $U(S_1)$ which is a function of the intermolecular energies of the pair of solutes for which a free energy change is to be calculated.

A dual topology setup is simpler and more generally applicable than a single topology setup. However dual topology approaches suffer from a number of technical difficulties which are mainly related to the fact that if a solute does not have any intermolecular interaction with its surroundings, it can drift anywhere in the simulation box. This usually causes the free energy difference to converge very very slowly (in practice not at all). To overcome these difficulties, the dual topology technique implemented in ProtoMS constrains a pair of solutes to stay together by the introduction of dummy bond between the center of geometry of the two solutes. As this does not prove to be sufficient to avoid convergence issues, a soft-core non bonded energy function is also implemented. In essence, the function that computes the intermolecular energy of the solutes is modified such that when a solute is not fully interacting with its surroundings, it's Lennard-Jones and couloumbic energies are softened such that atomic overlaps do not result in very large, positive, energies. The solute is effectively 'softer'. The function implemented in ProtoMS for a solute that is being turned off is described by equation 3.20.

$$U_{nonbonded,\lambda} = (1-\lambda)4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}^{12}}{(\lambda\delta\sigma_{ij}+r_{ij}^2)^6}\right) - \left(\frac{\sigma_{ij}^{6}}{(\lambda\delta\sigma_{iJ}+r_{ij}^2)^3}\right)\right] + \frac{(1-\lambda)^n q_i q_j}{4\pi\varepsilon_0\sqrt{(\lambda+r_{ij}^2)}} \tag{3.20}$$

where the parameters n and δ control the softness of the Coulombic and Lennard-Jones interactions respectively.

Calculating Free Energies

ProtoMS can be used to calculate the relative free energy along the λ-coordinate using a variety of methods. The methods fall into two broad catagories; multivalue methods and singlevalue methods. Multivalue methods simulate the system using multiple, linked copies of the system, each at a different λ-value. One copy is known as the reference copy (or reference state). The simulation trajectory is evolved using the forcefield set to the value of λ for this copy. The other copies of the system, known as perturbed copies (or perturbed states) follow this trajectory, but at different λ-values. For example, the reference copy could be used to generate a trajectory at λ. The energy $E_\lambda(x)$ for each configuration, x, of this trajectory is evaluated. A perturbed copy could be run at $\lambda+\Delta\lambda$, and the energy, $E_{\lambda+\Delta\lambda}(x)$, for each configuration, x, generated by the reference copy is also evaluated. The free energy difference, $\Delta G = G(\lambda+\Delta\lambda) - G(\lambda)$ between these two values of λ can be calculated as an average over the difference between these two energies;

$$\Delta G(\lambda) = -k_B T \ln\left\langle \exp^{-(E_{\lambda+\Delta\lambda}(x)-E_\lambda(x))/k_B T}\right\rangle_\lambda, \tag{3.21}$$

where k_B is Boltzmann's constant, T is the simulation temperature and the angle brackets $\left\langle ...\right\rangle_\lambda$ represent an average over all of the configurations generated by the reference copy at λ. This is known as the Zwanzig equation[1] and it lies at the root of all of the multivalue free energy methods implemented in ProtoMS;

Free Energy Perturbation (FEP) Simulations are run with the reference copy at a range of values across λ, e.g. at λ-values 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0. Perturbed copies are set to the values of λ for neighbouring simulations, e.g. the simulation with a reference copy at $\lambda = 0.4$ has reference copies at $\lambda = 0.2$ and $\lambda = 0.6$. The simulations are run, and the free energy differences from each λ-value to the next (the forwards differences) are summed to yield the total free energy across λ, and the free energy differences from each λ-value to the previous λ-value (the backwards differences) are summed to yield the negative of the free energy across λ. If the simulation has converged then the sum of the forwards differences should be equal to the negative of the sum of the backwards differences. For more detail about FEP see appendix A.1.1.

Finite Difference Thermodynamic Integration (FDTI) Simulations are run with the reference copy at a range of values across λ, just as in FEP. In this case perturbed copies are run at $\lambda+\delta\lambda$ and $\lambda-\delta\lambda$. If δλ is very small, then the free energy differences divided by δλ ($\frac{-\Delta G_{\text{backwards}}}{\delta\lambda}$ and $\frac{\Delta G_{\text{forwards}}}{\delta\lambda}$) will be good estimates of the free energy gradient with respect to λ. The integral of these gradients across λ will be the total free energy difference. In my experience, a value of δλ is normally sufficiently small to converge the free energy

gradients. For more details about FDTI see appendix A.1.2.

Replica Exchange Thermodynamic Integration (RETI) RETI uses the same methodology as FDTI, except that trajectories can exchange their reference and perturbed states. For more details about RETI see appendix A.1.3.

As well as implementing multivalue free energy methods, ProtoMS also implements singlevalue methods. Singlevalue methods calculate the free energy using only a single copy of the system, at only a single λ-value. The singlevalue methods implemented in ProtoMS work by changing the value of λ while a simulation trajectory is being generated;

Slow Growth In this method, λ is slowly increased from 0.0 to 1.0 as the simulation trajectory is generated. The energy associated with each increase in λ is accumalated, and if the increases in λ were sufficiently small, such that the system was able to fully equilibrate between changes, then the sum of the change in energy will be a good estimate of the free energy change across λ. For more detail about slow growth, see appendix A.2.1.

Fast Growth The fast growth method is very similar to slow growth, and uses the same method to estimate the free energy change. However, the fast growth method recognises that if λ is changed quickly then the resulting free energy will have been calculated from a non-equilibrium trajectory, and thus it will include an element of work that has been performed on the system. The fast growth method uses the Jarzynski equality to average the results of many slow growth simulations to return a good estime of the change in free energy. Because the average removes the error associated with changing λ too quickly, it is possible to use the Jarzynski equality to average many fast simulations together. For more detail about fast growth, see appendix A.2.1.

Umbrella Sampling Umbrella sampling is an umbrella term that covers a wide variety of methods. When applied to free energy calculations, umbrella sampling refers to a method where moves are made along the λ-coordinate as well as the normal structural coordinates. By using a dynamic λ-coordinate, the system is free to explore the perturbation being studied. For this exploration to be useful, the simulation must be encouraged to sample the entire λ-coordinate. Umbrella potentials may be used to bias the sampling along different regions of λ. It is difficult to know what umbrella to use, so the adaptive umbrella sampling method generates an ideal umbrella potential via an iterative scheme. Once the iteration has converged, the umbrella potential should encourage even sampling of the entire λ-coordinate, at which point the umbrella must equal

the negative of the potential of mean force along λ. For more detail about umbrella sampling, see appendix A.2.2.

3.1.4 Generic Moves

ProtoMS conducts a simulation by performing a sequence of moves on the system. The following moves are currently implemented;

Residue moves Standard Monte Carlo (MC) moves on protein residues.

Solute moves Standard MC moves on solute molecules.

Solvent moves Standard MC moves on solvent molecule.

Volume moves Monte Carlo moves that change the volume of the system. These are used to run constant pressure simulations.

λ-moves Monte Carlo moves that change λ. These may be used to perform umbrella sampling free energy simulations.

Dual potential moves Works only with implicit solvent simulations. Allows to sample rapidly configurations with a crude potential but correct for errors with a specific acceptance test.

Residue Moves

A residue move is a Monte Carlo move on a single protein residue. Obviously, for a residue move to be be performed, at least one protein that has flexible residues must be loaded. Each residue move comprises the following steps;

1. A protein is picked randomly from the set of proteins that have flexible residues. Note that each protein is weighted equally, so each protein has an equal chance of being chosen, regardless of how many flexible residues it contains. This behaviour is likely to change in future versions of the code, as ideally the probability of choosing to move a protein should be proportional to the number of flexible residues.

2. One of the flexible residues within the protein is chosen randomly from the set of all flexible residues in the protein. Again, there is no weighting of residues, so each flexible residue has an even chance of being chosen, despite the size of each residue.

3. If the backbone of this residue is flexible, then a random number between 1 and 3 is generated. If the random number is equal to 1, then only a backbone move on the residue will be attempted (see figure 3.1). If the random number is equal to 2 then only a sidechain move will be attempted, where all of the flexible internals of the residue are moved. If the random number is equal to 3 then a backbone and sidechain move are attempted simultaneously. If the backbone of this residue is fixed, then only a sidechain move is attempted.

4. The change in energy that results from this move is evaluated, and then tested according to the Metropolis criterion (see appendix B) to decide whether or not to accept the move.

5. If the move is accepted, then the new configuration of the residue is saved. If the move was rejected then the original configuration of the residue is restored.

You can change the flexibility of any residue in any protein by using the `fixbackbone` and `fixresidues` commands described in section 3.6.5. All residues of all proteins are flexible by default, and have flexible backbones. Note that the backbone move is still experimental and not thouroughly tested. I recommend that you fix the backbone of all residues for production simulations.

You control the maximum amounts that the residue moves via the residue template (see section 3.8.2). The actual amount that a residue moves by will be based on random values generated within the limits of the maximum amounts set in the residue template, e.g. if the maximum change of an angle was 5.0 °, then the angle will be changed by a random value generated evenly between -5.0 ° and $+5.0$ °.

Solute Moves

A solute move is a Monte Carlo move on a single solute molecule. Obviously, for a solute move to be performed, at least one solute molecule must be loaded. Each solute move comprises the following steps;

1. A solute is picked randomly from the set of loaded solutes. Each solute is weighted equally, regardless of its size or numbers of degrees of freedom.

2. One of the residues is chosen at random within the solute. Again, each residue is weighted equally, regardless of its size.

3. All of the flexible internals of this residue are changed, and the whole solute molecule is randomly translated, and rotated around its center of geometry.

4. The change in energy associated with this move is evaluated and then tested via the Metropolis criterion (see appendix B) to decide whether or not to accept the move.

5. If the move is accepted then the new configuration of the solute is saved. If the move was rejected then the original configuration is restored.

You can control the maximum amounts that the solute moves via the solute template (see section 3.8.2).

Solvent Moves

A solvent move is a Monte Carlo move on a single solvent molecule. Obviously, for a solvent move to be performed, at least one solvent molecule must be loaded. Each solvent move comprises the following steps;

1. A solvent molecule is randomly chosed from the set of loaded solvent molecules. If preferential sampling is turned on (see sections 3.4 and appendix B), then the solvent molecules closest to the preferred solute have a relatively higher weight, so will be more likely to be chosen. If preferential sampling is off, then each solvent is weighted equally, regardless of its relative size or proximity to a solute.

2. The solvent molecule is randomly translated and rotated around its center of geometry.

3. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion if preferential sampling was turned off, or via a biased Monte Carlo test if preferential sampling were turned on (see appendex B).

4. If the move was accepted then the new solvent configuration is saved, otherwise the original configuration is restored.

You can control the maximum amounts that the solvent is translated and rotated by by editing its solvent template (see section 3.6.1).

Volume Moves

A volume move is a Monte Carlo move that changes the volume of the system. This is needed to be able to perform Monte Carlo simulations at constant pressure (i.e. using the NPT ensemble). For a volume move to be performed you need to have loaded a box of solvent molecules, and be running using periodic boundary conditions (see section 3.8.5). A volume move is comprised of the following steps;

1. A random change in volume is chosen within the range set via the `maxvolchange` command (see section 3.4).

2. The volume of the system is changed by this amount by scaling all of the coordinates evenly from the center of the simulation box.

3. The change in energy associated with this change in volume is evaluated and used to decide whether or not to accept this move via the constant pressure Monte Carlo test (see appendix B), for the system pressure set via the `pressure` command (see section 3.4).

4. If the move is accepted then the new system configuration is saved, otherwise the original system configuration is restored.

λ-moves

A λ-move is a Monte Carlo move that randomly changes the value of λ. This feature is in the process of being added to ProtoMS, and is not ready for use in the current version. λ-moves will be available in the next release of ProtoMS.

Relative Move Probabilities

You can specify which moves should be run by passing arguments to the `simulate` and `equilibrate` commands (see section 3.6.1). You can use these commands to assign a weight to each type of move, e.g. 100 for solvent moves, 10 for protein moves, 1 for solute moves and 0 for volume move. The type of move chosen for each step of the simulation is generated randomly based on these set relative weights. These weights mean that on average, in 111 moves, 100 of these moves will be solvent moves, 10 of these moves will be protein moves, 1 of these moves will be solute moves and none of the moves will be volume moves (e.g. no volume moves will be performed). Note that you need to perform some volume moves if you wish to sample from the NPT ensemble!

3.2 Interface

Like a lot of simulation programs, ProtoMS is not the easiest piece of software to use! ProtoMS is a simple program that may be used from the command line. Once you have compiled it you should find it in the top directory (it is called simply `protoms2`). If you run the program you should see that it prints out some information about the program and license, then it complains that nothing has been loaded so it closes down.

The interface to ProtoMS has been designed to allow easy integration of ProtoMS with scripts, and to enable simple use from a command file. A ProtoMS input consists of a set of commands and values, e.g. the command 'temperature' could have the value '25.0'. This would set the simulation temperature to 25°C. The input can be

passed to ProtoMS via an environmental variable or via a command file. The above command could thus be input by setting the environmental variable 'temperature' equal to '25.0', or by placing the line

```
temperature 25.0
```

into a file and have ProtoMS read commands from that file. You specify the command file by passing it to ProtoMS on the command line, e.g.

```
protoms2 mycmdfile.txt
```

If the command exists in the command file and as an environmental variable then the value held in the environmental variable is used. This allows you to write a generic command file that contains most of the description of your simulation, and then to change the environment variable in a script to run different simulations with changed input. Note that the ProtoMS is insensitive to whether commands, variables or contents of files are uppercase or lowercase, so you are free to mix and match capitals and small case wherever you want. The only exception to this is in the specification of filenames, where your operating system may care about case!

Higher level interfaces to ProtoMS are provided for the Perl and Python scripting languages. These interfaces are described in sections 3.7.1 and 3.7.2.

3.3 File Output

If you run ProtoMS from the command line you should see that it prints out a lot of information to the screen (on Unix called standard output, STDOUT). If you look closely at the output you should see that each line of output is preceeded by a tag, such as 'HEADER' or 'INFO'. ProtoMS uses streams to output data, and these tags state which stream the line of data came from. Thus the information at the top of the output that gives the license and version details has been printed to the 'HEADER' stream, while the lines stating that ProtoMS is closing down because nothing has been loaded have gone to the 'FATAL' stream. ProtoMS uses the following streams;

HEADER Used to print the program header.

INFO Used to print general information.

WARNING Warnings are printed to this stream. ProtoMS will generally try to continue if it detects a problem, and will print out information about any errors to the WARNING stream. It is up to you to check the WARNING stream to ensure that your simulation is working correctly.

FATAL If an error is so serious that ProtoMS is forced to shutdown then it will first try to tell you what the

problem is by sending text to the FATAL stream.

RESTART The restart file is written to the RESTART stream.

PDB Any output PDB files are written to the PDB stream.

MOVE Information about moves are printed to this stream, e.g. whether or not a move was accepted, and how much progress has been made during the simulation.

ENERGY Information about the energy components for the moves are printed to this stream, e.g. the bond energy of solute 1, or the coulomb energy between protein 1 and the solvent.

RESULTS The results of the simulation are written to the RESULTS stream. These include the free energy averages and energy component averages.

DETAIL The DETAIL stream contains lots of additional detail about the setup of the simulation. This can be very verbose, as it includes complete detail of the connectivity of the system and the loaded forcefield. The DETAIL stream is useful when you are setting a simulation up, though should be turned off when you are running production.

SPENERGY The SPENERGY stream is used to report the results of single point energy calculations.

ACCEPT The ACCEPT stream is used to print information about the numbers of attempted and accepted moves.

RETI The RETI stream is used to report the energies needed by the RETI free energy method.

DEBUG The DEBUG stream is used by the developers to report debugging information during a ProtoMS run. This stream is only active if ‘debug’ is set to true.

These streams may be switched on or off, directed to STDOUT, directed to STDERR or directed to a file. You can do this by using the commands

```
streamSTREAM STDOUT
```

```
streamSTREAM STDERR
```

```
streamSTREAM off
```

```
streamSTREAM /path/to/file.txt
```

where `STREAM` is the name of the stream that you wish to direct (e.g. streamINFO). ProtoMS is insensitive to case, so you could use the command

```
streaminfo stdout
```

However, your operating system may be sensitive to case so you should ensure that you use the correct case for filenames.

You are free to direct multiple streams into a single file, or to turn undesired streams off. If a stream is output to STDOUT or STDERR then the name of the stream is prepended to the start of each line. The name is not attached if the stream is directed into a file. The WARNING and FATAL streams are special as unlike the other streams, these two cannot be turned off. These two streams will be directed to STDERR if they have not been directed elsewhere.

By default, the HEADER, INFO, MOVE and RESULTS streams are directed to STDOUT, the WARNING and FATAL streams are directed to STDERR, and the remaining streams are switched off. Bear this in mind if you think that you should be getting output and you are not - make sure that the stream that contains your output is directed to something!

The streamSTREAM command is used to specify the direction of the stream at the start of the simulation. It is possible to redirect streams while the simulation is running. This is slightly more complicated than then streamSTREAM command, and is described in section 3.6.

3.4 Simulation Parameters

There are many commands to set parameters that you can use to control your simulation. These are;

```
debug logical
```

where *`logical`* is `true` or `false`, `yes` or `no`, `on` or `off` (depending on your personal preference). This turns on or off debugging output that may be useful for ProtoMS developers. By default `debug` is `off`.

```
testenergy logical
```

where *`logical`* has the same values as for `debug`. This is used to set whether or not to turn on testing of energies. This is useful if you are developing ProtoMS. By default `testenergy` is `off`.

```
prettyprint logical
```

Turn on or off pretty printing. With pretty printing turned on, you will see nice starry boxes drawn highlighting certain parts of the output. By default, `prettyprint` is `on`.

```
dryrun logical
```

Whether or not to perform a dry run of the simulation. If this is true then all of the files will be loaded up and your commands parsed. If there are any problems then these will be reported in the WARNING stream. No actual simulation will be run, though any files that would be created *may* be created. While this option is very useful for testing your commands, it is not perfect and cannot check everything. I thus recommend that you also perform a short version of your simulation before you commit yourself to full production. By default `dryrun` is `off`.

```
ranseed integer
```

where *integer* is any positive integer. This command is used to set the random number seed to be used by the random number generator. The random number seed can be any positive integer, and you will want to specify a seed if you wish to run reproducable simulations. If you don't specify a random number seed then a seed is generated based on the time and date that the simulation started.

```
temperature float
```

where *float* is any floating point number. Use this command to specify the simulation temperature in celsius. By default `temperature` is 25.0 °C.

```
lambda float
```

where *float* is a number between 0.0 and 1.0. Specify the value of λ. If a single value is given then that is used for λ. If three values are given then these are used for λ, and λ in the forwards and backwards windows, e.g.

```
lambda 0.5 0.6 0.4
```

would set λ for the reference state to 0.5, λ for the forwards perturbed state to 0.6, and λ for the backwards perturbed state to 0.4. By default all values of λ are 0.0. Section 3.6 describes the method used to change the values of λ during a running simulation.

```
cutoff float
```

where *float* is any positive number. This command is used to set the size of the non-bonded cutoff, in Angstroms, used to truncate the intermolecular non-bonded potentials (see equation 3.2). By default the non-bonded cutoff is 15Å.

```
feather float
```

To prevent an abrupt cutoff, the non-bonded energy is scaled quadratically down to zero over the last part of the cutoff (see equation 3.2). The feather command sets the distance over which this scaling occurs, e.g.

```
feather 1.3
```

sets this feathering to occur over the last 1.3Å. The default value of the feather is 0.5Å.

```
cuttype type
```

where *type* is either `residue` or `molecule`. This specifies the type of non-bonded cutting to use; either `residue`, where the cutoff is between protein residues, solute molecules and solvent molecules, or `molecule`, where the cutoff is between protein molecules, solutes molecules and solvent molecules. This was described in more detail in section 3.1.2. By default the cuttype is `residue`.

```
pressure float
```

This command sets the pressure of the system in atmospheres. By setting the pressure to a non-zero value you will be able to perform a simulation in the NPT isothermal-isobaric ensemble. Note that you need to perform volume moves (see section 3.1.4) to be able to run in the NPT ensemble. By default the pressure is equal to zero, and thus a NPT simulation is not performed.

```
maxvolchange float
```

This command sets the maximum change in volume for a volume move in cubic Angstroms. This command only has meaning if an NPT simulation is being performed. By default maxvolchange is equal to the number of solvent molecules divided by ten.

```
prefsampling integer
```

This command is used to turn on preferential sampling of the solvent (see section 3.1.4), and to specify which solute is used to define the center of the preferential sampling sphere. The command

```
prefsampling 1
```

means that the solvents closest to solute 1 will be moved more frequently than those furthest from solute 1. An optional parameter may be used to change the influence of the sphere, e.g.

```
prefsampling 1 100.0
```

will specify a preferential sampling sphere centered on solute 1, with a parameter of 100.0. The larger the parameter, the more highly focussed the influence of the sphere around the closest solvent molecules. By default the parameter is 200.0, and preferential sampling is turned off.

```
dualtopology int int1 int2 synctrans syncrot
```

This turns on the dual topology method of calculating relative free energies, where *int1* is the perturbed solute at $\lambda = 0.0$ and *in2* is the solute at $\lambda = 1.0$. If synctrans is set, the rigid body translations of the two solutes will be synchronised. If syncrot is set, the rigid body rotations of the two solutes will also be synchronised.

```
boundary none
```

This turns off any boundary conditions, i.e. the simulation will be performed in vacuum.

```
boundary periodic dimx dimy dimz
```

This turns on periodic boundaries, using a orthorhombic box centered on the origin, with dimensions *dimx* Å by *dimy* Å by *dimz* Å. Note that these dimensions may be modified by any loaded solvent file (see section 3.8.5).

```
boundary periodic ox oy oz tx ty tz
```

This turns on periodic boundaries using an orthorhombic box with the bottom-left-back corner at coordinates (*ox*,*oy*,*oz*) Å and the top-right-front corner at (*tx*,*ty*,*tz*) Å. Note that these dimensions may be modified by any loaded solvent file (see section 3.8.5).

```
boundary cap ox oy oz rad k
```

This turns on solvent cap boundary conditions. Protein and solute molecules will experience no boundary conditions, while solvent molecules will be restrained within a spherical region of radius *rad* Å, centered at coordinates (*ox*,*oy*,*oz*) Å. A half-harmonic restraint with force constant *k* kcal mol^{-1} Å $^{-2}$ is added to the solvent energy if it moves outside of this sphere.

```
boundary solvent
```

This sets the boundary conditions to whatever is set by the loaded solvent files. If no solvent files are loaded then no boundary conditions are used. This is the default option, and the method of setting boundary conditions via a solvent file is described in section 3.8.5. softcore]

```
softcore int solute int
```

This causes the intermolecular energy of solute *int* to be softened. Alternatively, you can write 'all' instead of the solute index and all solutes will have their non bonded energy softened. The softcore is only supported for solutes. softcoreparams]

```
softcoreparams coul 0 delta 1.5 gb 0
```

This causes the solutes non bonded energy to be softened with a parameter n set to 0 and δ set to 1.5. (see eq 3.20). If conducting a GBSA simulation, this also causes the GB energy to be softened as well. It is recommended to use the same parameter for the Coulombic and Generalised Born energy. The values listed here, seem to work well for a number of relative binding free energy calculations but actual optimum values of these parameters will depend on your system. surface]

```
surface quality 3 probe 1.4
```

This command will cause surface area calculations to be performed during the simulation. quality can be set to 1,2,3,4 and will result in increasingly precise surface area calculations. For typical simulations, 3 should be fine and 2 will not give a huge error. Probe is the radius of the probe and should be set to 1.4 if you want to calculate the solvent accessible surface area of water, but can be set to 0 if you want to calculate the van der waals surface area of a molecule. born]

```
bor cut 20 threshold 0.005 proteins
```

This command will enable Generalised Born energy calculations. Thus to run a full GBSA simulation you should use both the surface and born keywords. cut controls the cutoff distance for the computation of the Born radii. If you work with a medium sized protein scoop of circa 100-150 residues, 20 should be fine but you may want a larger value for simulations of large proteins. threshold controls the number of pairwise terms that are not updated when the effective Born radii must be calculated by the Pairwise descreening approximation. The default value 0.005 appear to be a good tradeoff. Increasing it will make the simulation faster but less accurate. proteins activates the rescaling of the Born radii to compensate for systematic errors of the Pairwise Descreening Approximation in large biomolecules. It should be used only when simulating proteins and then its effectiveness has not been yet convincingly demonstrated.

3.5 Specifying Input Files

As well as controlling the simulation, commands are also used to specify the names of the input files that describe the system and forcefield for the simulation. These input files are specified using the following commands;

```
proteinN filename
```

Specifies the name of the Nth protein file, e.g.

```
protein1 protein.pdb
```

specifies that protein 1 should be loaded from the file `protein.pdb`. Note that proteins must be numbered sequentially from 1 to MAXPROTEINS. The format of a protein file is described in section 3.8.3.

```
soluteN filename
```

Specifies the name of the Nth solute file. Note that the solutes must be numbered sequentially from 1 to MAXSOLUTES. The format of a solute file is described in section 3.8.4.

```
solventN filename
```

Specifies the name of the Nth solvent file. Unlike the protein and solute files, the solvent file may contain multiple solvent molecules, though the total number of solvent molecules cannot exceed MAXSOLVENTS. The format of a solvent file is described in section 3.8.5.

```
parfileN filename
```

Specify the name of the Nth parameter file. You can specify as many parameter files as you wish, as long as you number them sequentially from 1 upwards. Each parameter file is read in sequence. If a parameter is repeated in a later parameter file then the parameter is overwritten. The last read value of a parameter is the value used in the simulation. The format of the parameter file is described in section 3.8.1.

The usage of these commands can be seen in the examples in chapter **??**.

3.6 Running a Simulation

A simulation is run as a sequence of chunks. Different things may be accomplished in each chunk, e.g. running some steps of equilibration, printing the protein coordinates to a PDB or redirecting a stream to a new file. Chunks may be mixed and matched, and you can run as many chunks as you desire within a single simulation. You specify a chunk using the command

```
chunkN chunk command
```

where N is the number of the chunk, e.g. `chunk1`. Chunks must be numbered sequentially from 1 upwards.

3.6.1 Equilibration and Production

The meat of a simulation is equilibration and production. In ProtoMS equilibration is defined as sampling without the collection of free energy or energy averages, while production is sampling with the collection of free energy and energy averages. Equilibration and production are specified using the `equilibrate` and `simulate` chunks, e.g.

```
chunk1 equilibrate 50
```

performs 50 steps of equilibration.

```
chunk2 simulate 1000
```

performs 1000 steps of production.

Additional options may be passed to these two chunks to control the probability of different types of move and the frequency of printing out move and energy details to the MOVE and ENERGY streams. These options are;

```
printmove=N
```

Print move and energy information every N moves.

```
protein=N
```

Set the relative probability of protein moves to N.

```
solute=N
```

Set the relative probability of solute moves to N.

```
solvent=N
```

Set the relative probability of solvent moves to N.

```
titrate=N
```

Set the relative probability of titration moves to N.

```
volume=N
```

Set the relative probability of volume moves to N.

```
lambda=N
```

Set the relative probability of λ-moves to N.

```
newprob
```

Reset relative move probabilities to zero.

Note that succeeding equilibration or production chunks inherit the move probabilities and printing frequency of preceeding simulation or equilibration chunks. I thus recommend that you use the `newprob` option to reset the move probabilities for each equilibration or production chunk you run.

The following examples illustrate the use of these options;

```
chunk1 newprob equilibrate 500 printmove=10 protein=1 solvent=1000
```

Perform 500 steps of equilibration, printing move and energy information every 10 moves, making on average 1 protein move for every 1000 solvent moves (and performing no other types of move).

```
chunk2 equilibrate 100 solute=500
```

Perform 100 steps of equilibration. Because this chunk will inherit from chunk1, the move and energy information will still be printed every 10 moves, and still, on average 1 protein move will be made every 1000 solvent moves. However this line has added that on average 500 solute moves should be made for every 1000 solvent moves, thus the probability of a protein move is now 1 in 1501, the probablity of a solute move is 500 in 1501, and the probability of a solvent move is 1000 in 1501.

```
chunk3 simulate 500 printmove=1 newprob volume=1 solvent=300
```

Now perform 500 steps of production, printing move and energy information every move, performing no protein moves, and 1 volume move for every 300 solvent moves.

```
chunkN splitgbsasimulate 100 10 solute=1 protein=9
```

The above command should only be used if you are doing an implicit solvent simulation (e.g, you turned on the surface and born keywords). This will cause to run 10 moves with a crude GBSA potential and then perform an acceptance test based on the difference of energies between the crude GBSA potential and the GBSA potential you set with the cutoff, born and surface keywords. This will be repeated 100 times. Here the move probabilities were set to 1 and 9 for solute and protein, but could be other figures. After this keyword has been used it is advised to use the following keyword.

```
chunkN resetgb
```

This will cause the total energy of the system to be calculated fully and the Born radii to be correctly updated. Periodic usage of this command, along with the previous one, avoids drifts in the total energy of the system.

3.6.2 Results and Restarts

As well as controlling the sampling, you can also control the collection and output of results using simulation chunks, and the reading and writing of restart files.

```
chunk1 averages reset
```

Reset all averages to zero and start collection of results from scratch.

```
chunk2 averages write
```

Write out the energy and free energy averages to the RESULTS stream. It is probably a good idea to do this a some point before the end of the simulation!

```
chunk3 averages write myfile.txt
```

Does the same as above, but redirects the RESULTS stream to myfile.txt before the results are written.

```
chunk4 restart write
```

Write a restart file for the current configuration to the RESTART stream.

```
chunk5 restart write myfile.txt
```

Does the same as above, but redirects the RESTART stream to myfile.txt before the restart file is written.

```
chunk6 restart read myfile.txt
```

Read in a restart file from the file myfile.txt.

3.6.3 PDB Output

You can use a simulation chunk to output a PDB of the current configuration. The output can be tailored to include only the parts of the system that you are interested in. This is useful if you are trying to conserve disk usage. You can output PDBs using the ‘pdb’ chunk;

```
chunk1 pdb all
```

Output a PDB of all proteins and solutes to the PDB stream

```
chunk2 pdb protein=all
```

Output a PDB of all proteins to the PDB stream

```
chunk3 pdb protein=2
```

Output a PDB of protein 2 to the PDB stream

```
chunk4 pdb solute=all
```

Output a PDB of all solutes to the PDB stream

```
chunk5 pdb solute=1
```

Output a PDB of solute 1 to the PDB stream

The output PDB can be controlled via additional commands added to the above lines, e.g.

```
chunk1 pdb all solvent=all
```

Output the PDB including all solvent molecules.

```
chunk2 pdb solute=1 solvent=5.0
```

Output a PDB including all solvent molecules within 5.0Å of whatever else is printed - in this case solute 1.

```
chunk3 pdb protein=1 showdummies
```

Output a PDB that also includes dummy atoms.

```
chunk4 pdb solute=all showhidden
```

Output a PDB that also includes hidden solute molecules (solutes that are used to perform geometry perturbations).

```
chunk5 pdb all file=myfile.txt
```

Redirect the PDB stream to myfile.txt then print the PDB.

3.6.4 Restraints

ProtoMS supports a number of restraining potentials which can be used to modify the potential energy function and bias the simulation towards particular configurations. To use a restraint in ProtoMS you must first assign an id number to a particular atom or set of atoms, using the following command

```
chunk1 id add int1 type int2 atname resname|resnumber
```

where int1 is the index numberr for this id. So if this if the first id you create you may want to use the number 1. type can be SOLUTE or SOLVENT or PROTEIN depending on where the atom you want to tagg is. atname is the name of the atom (e.g CA), resname is the name of the residue the atom is in if you are dealing with a SOLUTE or SOLVENT. However if it the atom is in a protein, then you must use the PDB residue number.

Once you have specified a few ids, you can create restraints using these ids and the following command

```
restraint add id1[-id2-id3-id4] type1 type2 [other parameters]
```

where id1 to id4 designate up to four ids. type 1 designate the type of the restraint. It can be either cartesian,bond or dihedral. In the first case the restraint is applied in cartesian coordinates and will apply to only one atom (id1). In the second case, it is applied in internal coordinates, and will apply to only two atoms (id1-id2). In the last case it is applied to four atoms (id1-id2-id3-id4) and in internal coordinates. type2 designate the functional form of the restraint. It can be harmonic or flatbottom. Each functional form requires additional parameters. The following options are currently possible:

```
restraint add id1 cartesian harmonic xrest yrest zrest krest
```

For a cartesian harmonic restraint you need to specify the coordinates of the anchoring point and the value of the force constant.

```
restraint add id1 cartesian harmonic xrest yrest zrest krest wrest
```

For a flatbottom restraint you must in addition specify the width of the flat region of the potential.

```
restraint add id1-id2 bond harmonic krest
```

For a bond restraint you must specify only the force constant

```
restraint add id1-id2-id3-id4 dihedral harmonic theta krest
```

For a dihedral harmonic restraint you must specify the target equilibrium angle and the force constant. This restraint does not work on solvent molecules and on protein backbone atoms.

The following example shows how to add a harmonic potential restraint between a ligand atom and a protein atom.

```
chunk1 id add 1 SOLUTE 1 N2 LI8
```

This chunk will create id number 1 which will point to solute atom 1 (the first atom in the solute pdb file), named

c00, from residue L10.

```
chunk2 id add 2 PROTEIN 1 O 318
```

This chunk will create id number 2 which will point to protein pdb loaded as protein1 by ProtoMS. The atom named O in residue 318 will be selected. Note that 318 is the residue number that appear in the PDB file. It is not necessarily the 318th residue to be loaded by ProtoMS in this protein.

```
restraint add 1-2 bond harmonic 5.0 3.33
```

This chunk will cause a restraint to be added between the atoms id 1 and 2 points to. The functional form of this restraint will be a harmonic potential that is function of the distance between these two atoms. The force constant will be 5 kcal mol^{-1}.angstrom-2 and the equilibrium distance 3.33 angstrom.

3.6.5 Miscellaneous

As well as running the simulation, there are also a collection of other things that you can do in a simulation chunk. These are;

```
chunk1 singlepoint
```

Calculate the energy of the current system and output it to the SPENERGY stream. This is useful if you just want to use ProtoMS to evaluate a forcefield energy. You can set up the input files, turn off all streams, direct stream SPENERGY to STDOUT and run a simulation that only consists of this 'singlepoint' chunk.

```
chunk2 soluteenergy N
```

Calculate the energy of solute N. This calculates the energy of solute N and outputs the components of this energy in great detail. This is useful for debugging a forcefield or for collecting average energy components that are more finely divided than those normally collected.

```
chunk3 retienergy 0.2
```

The RETI free energy method requires the calculation of the energy at the neighbouring two λ windows at the end of the simulation. This chunk will calculate the energy at λ windows 0.2 above and below the reference state, and will output the results to the RETI stream.

```
chunk4 lambda 0.5
```

Sets λ to 0.5. Will calculate and return the change in energy associated with this change in λ. This is useful if

you wish to perform a slow growth or fast growth free energy simulation. You could also use this in conjunction with the 'averages print' and 'averages reset' chunks to calculate the free energy of all windows across λ within a single simulation. This is because the window widths are preserved by the change in λ, thus if the λ windows were 0.1 0.2 0.4 before the change, then they would be 0.4 0.5 0.7 after the change. Note that the values of λ are clamped between 0.0 and 1.0.

```
chunk5 lambda 0.5 0.6 0.4
```

As above, except set the λ values of the forwards and backwards windows to 0.6 and 0.4 respectively.

```
chunk6 lambda delta 0.1
```

As above except instead of directly setting λ, change λ by 0.1. This will also increase the value of λ for the forwards and backwards windows by 0.1.

```
chunk8 fixresidues 1 all
```

Fix all of the residues of protein 1.

```
chunk8 fixresidues 1 1-10 12 14 16-20
```

Fix the residues of protein 1. Only fix residues 1 to 10, 12, 14 and 16 to 20.

```
chunk9 fixresidues 1 none
```

Unfix all of the residues of protein 1.

```
chunk10 fixbackbone 1 all
```

Fix the backbone of all residues of protein 1. This chunk has the same syntax as the fixresidues chunk.

```
chunk11 fixbackbone 1 none 20-35
```

Unfix all of the residues of protein 1, then fix the backbone of residues 20-35. This ensures that only the backbone of residues 20-35 is fixed.

```
setstream info=stdout move=off
```

Direct the INFO stream to STDOUT and turn the MOVE stream off.

```
setstream restart=myfile.txt warning=stderr
```

Direct the RESTART stream to myfile.txt and the WARNING stream to STDERR. solvate]

```
chunkN solvent box xdim ydim zdim [xorig yorig zorig xmax ymax zmax]
```

This command can be used to replicate a solvent file loaded as solvent1 such that the final solvent occupies a box of dimensions xdim ydim zdim with origin (0,0,0). Alternatively the origin can be specified along with the maximum coordinates of the cubix box. solvate2]

```
chunkN solvent cap xorig yorig zorig rad
```

As before but the output will be a spherical cap of solvent centered at the specified origin and with a radius rad.

The last two commands can be used to create large solvent boxes when needed. Once this chunk has been performed, you should save a pdb of the system using the chunk pdb and then edit the output file such that it can load as ProtoMS solvent pdb. The process of replicating the solvent molecules can be quite memory consuming and you may find you have to recompile ProtoMS so that it can handle a large number of solvent molecules, particularly if the coordinates of the system you want to solvate are far away from the coordinates of the solvent molecules in the input solvent box.

3.7 Scripting Language Interfaces

As well as being able to be used via the command line or via a command file, it is also possible to embed ProtoMS within a script. To help you, interfaces between ProtoMS and two common scripting languages have been provided (Perl and Python). These interfaces have written to be very similar to each other, so that you should be able to use them in an almost indentical manner within your scripts. Both interfaces work by generating an internal dictionary of all of the commands and values (see section 3.2). This can then either be turned into a set of environmental variables that are fed to ProtoMS when it is run, or it can be written out as a ProtoMS command file.

Both of the interfaces have deliberately been kept simple. I encourage you to look at the code for these interfaces and use them as a foundation for your own, more sophisticated versions!

3.7.1 Python Interface

The Python module is provided in the file `protoms.py`, located in the `interfaces` directory. This interface is known to work with Python version 2.3. Figure 3.3 shows an example of importing this module into your Python script, and using it to set up a basic simulation.

The Python interface provides a `Simulation` object that is used to set all of the simulation parameters and description. The functions available for this object are;

```
s = protoms.Simulation()
```

```
#!/bin/env python

# Set the module search path to include the ProtoMS interfaces directory
# (you will need to modify this path
import sys
sys.path.append("/path/to/ProtoMS/interfaces/directory/")
# import the protoms.py module
import protoms

# Everything is handled via a Simulation object
s = protoms.Simulation()

# add the file for protein 1
s.setProtein(1,"/path/to/protein.pdb")
# add the file for solute 1
s.setSolute(1,"/path/to/solute.pdb")
# add the solvent
s.setSolvent(1,"/path/to/solvent.pdb")

# set the temperature to 20 C
s.setParameter("temperature",20.0)

#set the locations of the output files
s.setStream("info","./output")
s.setStream("header","./output")

#use n to keep track of which chunk we are on
n = 1

#now run 10 blocks of simulation, writing a PDB after each block
for i in range(0,10):
      #perform 1000 steps of simulation (this returns n+1)
      n = s.setChunk(n, "simulate 1000")
      #print a pdb to the file 'output-i.pdb', e.g. output-1.pdb
      n = s.setChunk(n, "pdb all solvent='5.0' file=output-%d.pdb" % i+1)

#finally print out the averages
n = s.setChunk(n,"averages write results.txt")

#write out this simulation to a command file
s.writeCommandFile("/path/to/command/file")

#now run the simulation
exitval = s.run("/path/to/protoms2-executable","/path/to/command/file")

if (exitval == 0):
      print "Simulation completed successfully!"
else:
      print "Something went wrong?"
```

Figure 3.3: An example Python script that imports the `protoms.py` interface and uses it to build a small simulation.

This constructs a new, empty, `Simulation` object.

```
s = protomc.Simulation("filename")
```

This constructs a new `Simulation` object, and fills it with the commands present in the specified ProtoMS command file.

```
s.clear()
```

This is used to clear the `Simulation` object of all commands.

```
s.readCommandFile("filename")
```

This will read in the commands present in the specified ProtoMS command file. Note that this will not unset the values of commands present in the `Simulation` object that are not in the command file. If you wish to ensure that the `Simulation` object only contains commands from the file, then you must `clear` it first.

```
s.setStream("stream","output")
```

This sets the contents of stream *stream* to be sent to *output* (see section 3.3).

```
s.setParameter("parameter","value")
```

This sets the parameter *parameter* to the value of *value* (see section 3.4).

```
s.setChunk(n,"chunk")
```

This sets chunk n to be equal to *chunk* (see section 3.6). This function returns $n+1$.

```
s.setForceField(n,"filename")
```

This sets the filename of forcefield file n. This function returns $n+1$.

```
s.setProtein(n,"filename")
```

This sets the filename of protein file n. This function returns $n+1$.

```
s.setSolute(n,"filename")
```

This sets the filename of solute file n. This function returns $n+1$.

```
s.setSolvent(n,"filename")
```

This sets the filename of solvent file n. This function returns $n+1$.

```
s.writeCommandFile("filename")
```

This writes the `Simulation` object out to a ProtoMS command file. This would be necessary if your simulation contains a lot of simulation chunks, as your operating system will limit the number of environmental variables that may be set for ProtoMS.

```
s.run("exefile")
```

Run the ProtoMS executable *exefile* using the simulation description held by this `Simulation` object. This function blocks until ProtoMS has finished, and returns the exit value.

```
s.run("exefile","cmdfile")
```

This function runs the ProtoMS executable *exefile* using the ProtoMS command file *cmdfile*. Note that this function will not use any of the information stored in the `Simulation` object, as it is provided as a convienient function to use with `writeCommandFile()`.

3.7.2 Perl Interface

The Perl interface is provided in the file `protomc.pm` located in the `interfaces` directory. This interface is known to work with Perl 5.4 and above (although probably not with the upcoming Perl 6). The Perl interface is used in an almost identical manner as the Python interface. The equivalent Perl version of the example shown in figure 3.3 is shown in figure 3.4.

The Perl interface provides a `Simulation` object that is used to set all of the simulation parameters and description. The functions available for this object are almost identical to those provided by the Python `Simulation` object, and are;

```
$s = Simulation::new();
```

This constructs a new, empty, `Simulation` object.

```
$s = Simulation::new("filename");
```

This constructs a new `Simulation` object, and fills it with the commands present in the specified ProtoMS command file.

```
$s->clear();
```

This is used to clear the `Simulation` object of all commands.

```
$s->readCommandFile("filename");
```

```
#!/bin/env perl

#Set the perl library search path to include the ProtoMS2 interfaces directory
use lib "/path/to/ProtoMS/interfaces/directory";
use protoms;

# Everything is handled via a Simulation object
$s = Simulation::new();

# add the file for protein 1
$s->setProtein(1,"/path/to/protein.pdb");
# add the file for solute 1
$s->setSolute(1,"/path/to/solute.pdb");
# add the solvent
$s->setSolvent(1,"/path/to/solvent.pdb");

# set the temperature to 20 C
$s->setParameter("temperature",20.0);

#set the locations of the output files
$s->setStream("info","./output");
$s->setStream("header","./output");

#use $n to keep track of which chunk we are on
$n = 1

#now run 10 blocks of simulation, writing a PDB after each block
for ($i=1; $i<=10; $i++)
{
      #perform 1000 steps of simulation
      $n = $s->setChunk($n, "simulate 1000");
      #print a pdb to the file 'output-i.pdb', e.g. output-1.pdb
      $n = $s->setChunk($n, "pdb all solvent='5.0' file=output-$i.pdb");
}

#finally print out the averages
$n = $s->setChunk($n,"averages write results.txt");

#write out this simulation to a command file
$s->writeCommandFile("/path/to/command/file");

#now run the simulation
$exitval = $s->run("/path/to/protoms2-executable","/path/to/command/file");

if ($exitval == 0)
{
      print "Simulation completed successfully!\n";
}
else
{
      print "Something went wrong?\n";
}
```

Figure 3.4: An example Perl script that imports the `protoms.pm` interface and uses it to build a small simulation. This script is equivalent to the Python script shown in figure 3.3.

This will read in the commands present in the specified ProtoMS command file. Note that this will not unset the values of commands present in the `Simulation` object that are not in the command file. If you wish to ensure that the `Simulation` object only contains commands from the file, then you must `clear` it first.

```
$s->setStream("stream","output");
```

This sets the contents of stream *stream* to be sent to *output* (see section 3.3).

```
$s->setParameter("parameter","value");
```

This sets the parameter *parameter* to the value of *value* (see section 3.4).

```
$s->setChunk(n,"chunk");
```

This sets chunk *n* to be equal to *chunk* (see section 3.6). This function returns $n+1$.

```
$s->setForceField(n,"filename");
```

This sets the filename of forcefield file *n*. This function returns $n+1$.

```
$s->setProtein(n,"filename");
```

This sets the filename of protein file *n*. This function returns $n+1$.

```
$s->setSolute(n,"filename");
```

This sets the filename of solute file *n*. This function returns $n+1$.

```
$s->setSolvent(n,"filename");
```

This sets the filename of solvent file *n*. This function returns $n+1$.

```
$s->writeCommandFile("filename");
```

This writes the `Simulation` object out to a ProtoMS command file. This would be necessary if your simulation contains a lot of simulation chunks, as your operating system will limit the number of environmental variables that may be set for ProtoMS.

```
$s->run("exefile");
```

Run the ProtoMS executable *exefile* using the simulation description held by this `Simulation` object. This function blocks until ProtoMS has finished, and returns the exit value.

```
$s->run("exefile","cmdfile");
```

This function runs the ProtoMS executable *exefile* using the ProtoMS command file *cmdfile*. Note that this function will not use any of the information stored in the `Simulation` object, as it is provided as a convienient function to use with `writeCommandFile()`.

3.8 Input Files

ProtoMS can read in five types of input file;

Parameter / Forcefield file These provide the forcefield parameters used in a simulation, and the templates (z-matricies) that are used to specify the connectivity and flexibility of the simulated molecules.

Protein file These are simple PDB format files that contain the coordinates of the protein chains to be simulated. Only one protein chain may be contained within each protein PDB file.

Solute file These are simple PDB format files that contain the coordinates of the solutes to be simulated. Only one solute may be contained within each solute PDB file.

Solvent file These are simple PDB format files that contain the coordinates of the solvent molecules to be simulated. Multiple solvent molecules may be contained within each solvent file.

Restart file These are files used by ProtoMS to save and restore the coordinates of all of the molecules in the system.

ProtoMS is insensitive to case, so you can mix upper case and lower case within these files without affecting how they are read.

3.8.1 Parameter / Forcefield Files

The parameter file is the most powerful, and hence the most complicated of all of the input files read by ProtoMS. The parameter file provides all of the forcefield parameters that are used in a simulation, and it also provides all of the templates that provide the connectivity and z-matrices of all of the loaded molecules. The parameter file uses a word based format, meaning that you can leave as many spaces between words on a line as you like, and you do not have to worry about lining up data into particular columns.

The general format of a parameter file is shown in figure 3.5. How ProtoMS reads the parameter file is controlled by which `mode` the file has been set. There are several different modes, and as figure 3.5 shows, it is possible to change between modes within a single file. The different modes are;

```
# comment lines start with a '#'

mode clj
#.... charge / Lennard Jones forcefield parameters

mode bond
#.... bond parameters

mode template
#.... templates

#parameter file uses a word-based format, so leave as many spaces as
#you want between words, e.g.
       mode            clj

mode bond   #comments can also go at the end of any lines, like this!

MoDe DiHeDrAl  # you can use whatever case you want (though try to make
               # things readable!
```

Figure 3.5: An example ProtoMS parameter file.

info This mode is used to read in control information for the forcefield.

clj This mode is used to read in the charge and Lennard Jones (clj) parameters for the simulation.

bond This mode is used to read in the bond parameters for the simulation.

angle This mode is used to read in the angle parameters.

ureybradley This mode is used to read in the Urey-Bradley parameters.

dihedral This mode is used to read in the dihedral parameters.

template This mode is used to read in the templates (z-matricies) used in the simulation. The template format is quite complex, so is described in the next section (section 3.8.2).

ProtoMS will only read lines that are valid within the mode that is being read. If ProtoMS could not read a line, or finds an incorrectly formatted line, then ProtoMS will print a message to the WARNING stream and will skip that line. It is therefore very important that you check the WARNING stream if you are writing or modifying a parameter file. To help you, ProtoMS will write out detailed information about a loaded parameter file to the DETAIL stream. You should check this output to ensure that any changes you make to a parameter file are being correctly loaded by ProtoMS.

ProtoMS can be asked to load as many forcefield files as you desire. Each parameter or template within the forcefield files has either a numerical or name based ID. If two forcefield files have parameters or templates that share the same ID, then ProtoMS will use the value that was read last. ProtoMS will of course warn you that it

has overwritten an earlier parameter (by outputting a message to the WARNING stream) but this behaviour could still trip you up! To help you, all of the parameters that use numerical IDs in the forcefield files supplied with ProtoMS use IDs that are between 1 and 2999. You can thus use numerical IDs that are greater than or equal to 3000 without worrying about a clash.

mode info

This mode is used to read in control information for the forcefield. This information is used to set parameters that affect which functions are used to evaluate the forcefield, and to set the values of forcefield-global parameters. The following lines are valid within this mode;

```
ljcombine type
```

where *type* can be `arithmetic` or `geometric`. This sets the combining rules used for the Lennard Jones σ parameter to either the arithmetic mean (as used by AMBER), or the geometric mean (as used by OPLS). See equations 3.4 and 3.5 for the functional forms of these combining rules.

```
scl14coul float
```

This sets the 1-4 coulombic scaling factor, e.g. for OPLS the value should be 0.5 (see equation 3.2).

```
scl14lj float
```

This sets the 1-4 Lennard Jones scaling factor, e.g. for OPLS the value should be 0.5 (see equation 3.2).

mode clj

This mode is used to read in the charge and Lennard Jones (clj) parameters used by the simulation (see equations 3.3, 3.11 and 3.12). Only one type of line is valid within this mode;

```
par id amber proton-number charge sigma epsilon
```

id is the unique identifying number for this clj parameter. This can be any number from 1 to MAXCLJ (by default this is 10000). If this ID is the same as an already read CLJ parameter, then ProtoMS will write a warning to the WARNING stream, and will overwrite the old CLJ parameter with the new parameter. To help prevent unintentional ID clashes, then the forcefields supplied with ProtoMS only use parameter IDs from 1 to 2000, and the solvent models supplied with ProtoMS use parameter IDs 2001 to 2999. You are thus free to use parameter IDs from 3000 in your own parameter files.

amber is the AMBER atom type associated with this clj parameter. The AMBER atom type is a two letter code that is used to identify the atom for the purposes of assigning bond, angle, dihedral or Urey-Bradley parameters. If this is a parameter for a dummy or non-chemical parameter, then the AMBER atom type should be '??'. Note that the AMBER type *is case sensitive*. This is different to other parts of ProtoMS, and is required as the GAFF forcefield uses case to distinguish between different AMBER types.

proton-number is the number of protons in the atom associated with this clj parameter, e.g. 1 for hydrogen, 6 for carbon or 8 for oxygen.

charge, *sigma* and *epsilon* are the partial charge (in $|e|$), and Lennard Jones σ (Å) and ε (kcal mol^{-1}) parameters associated with this clj parameter, e.g.

```
par 2001 OW 8 -0.834 3.15061 0.1521 # TIP3P oxygen
```

specifies the clj parameter for oxygen in TIP3P water, with parameter number 2001, AMBER atom type 'OW' proton number 8, a partial charge of -0.834 $|e|$, $\sigma = 3.15061$ Å and $\varepsilon = 0.1521$ kcal mol^{-1}.

Parameter ID 0 is a special clj parameter used to represent a null atom. This null atom has charge, σ and ε values of 0.0, an AMBER atom type of 'DM' and a proton number of 0.

mode bond

This mode is used to read in the bond parameters used by the simulation. Two types of line are valid within this mode;

```
par id force-constant bond-length
```

id is an identifying number from 1 to MAXBNDPARAM (default 5000) that is used to uniquely identify a bond. As in the case of the clj parameters, new parameters with the same ID number will overwrite old parameters with that ID number, and the parameter files supplied with ProtoMS will only use IDs from 1 to 2999, so you can safely use parameters 3000 and up.

force-constant is the force constant (k_{bond}, see equation 3.6) for the bond parameter. The units of k_{bond} are kcal mol^{-1} Å^{-2}. *bond-length* is the equilibrium bond length (r_0), in units of Å.

The second type of line valid in this mode is used to associate a pair of AMBER atom types with a bond parameter;

```
atm amb1 amb2 id
```

This line specifies the bond between atoms with AMBER atom types *amb1* and *amb2* is assigned the parameters

from bond ID *id*. Note that this bond parameter does not need to have been loaded when this line of the parameter file is being read, as bond parameters are not assigned until after all parameter files have been read. If none of the bond parameter files provide this bond ID, then ProtoMS will print a message to the WARNING stream and will set the bond ID to 0. As in the case of the clj parameters, 0 is a special parameter used to specify a null bond, whose bond parameters, and thus energy, are all 0.0. In addition, any bond involving an AMBER atom with a null clj parameter (i.e. having AMBER atom type 'DM') will be automatically set to use bond parameter 0. It is not possible to have a non-null bond parameter for bonds that involve dummy atoms.

These bond `atm` lines are indexed by the AMBER pair *amber1-amber2*. If this AMBER pair has already been loaded then its parameter is overwritten with the new parameter. Note that bonds are symmetrical, thus bond index *amb1-amb2* is equal to *amb2-amb1*.

mode angle

This mode is used to read in the angle parameters used in the simulation and its format and behaviour is almost identical to that used in the bond mode. Again, only two types of line are valid within the angle mode;

```
par id force-constant angle-size
```

and;

```
atm amb1 amb2 amb3 id
```

id is an indentifying number from 1 to MAXANGPARAM (default 5000) that is used to uniquely identify an angle parameters. *force-constant* is the force constant (k_{angle}, see equation 3.7) for the angle parameter, in units of kcal mol^{-1} degree^{-2}. *angle-size* is the equilibrium angle size (θ_0) in units of degrees. Angle ID 0 is the null angle, and the forcefield files supplied with ProtoMS will only use angle IDs from 1 to 2999.

The `atm` line is again very similar to that in the bond mode, with in this case the angle between atoms with AMBER types *amb1-amb2-amb3* being assigned angle parameter *id*. Angles are also symmetric, so *amb1-amb2-amb3* is equivalent to *amb3-amb2-amb1*. Like the bond mode, any angle involving dummy atoms (AMBER type 'DM') will automatically be set to use the angle parameter 0. It is not possible to use a non-null angle parameter over an angle involving dummy atoms.

mode ureybradley

This mode is used to read in Urey-Bradley parameters (see equation 3.8), and its format is identical to that of the angle mode. There are only two valid lines in this mode;

```
par id force-constant uby-size
```

and;

```
atm amb1 amb2 amb3 id
```

In this case *force-constant* refers to the Urey-Bradley force constant (k_{uby}), in units of kcal mol^{-1} Å $^{-2}$ and *uby-size* refers to the equilibrium Urey-Bradley length (x_0) in units of Å. Everything else about this mode is identical to that of the bond mode.

mode dihedral

This mode is used to read in the dihedral parameters that are used in the simulation. There are three types of line that are value in this mode. The first of these is used to provide the parameters for a single dihedral cosine term;

```
term term-id k1 k2 k3 k4
```

term-id is an ID number from 1 to MAXDIHTERMS (default 5000) that uniquely identifies this dihedral cosine term. *k1* to *k4* are the values of the four constants (k_1 to k_4) that control the dihedral cosine term (see equation 3.9). k_1 has units of kcal mol^{-1}, k_2 and k_3 are dimensionless, and k_4 is in units of degrees.

A full dihedral parameter is composed from the sum of individual dihedral cosine terms. The second valid line in the dihedral mode specifies which terms are associated with which parameters, e.g.

```
par id 3 10 32
```

specifies that dihedral parameter *id* is formed as the sum of dihedral cosine terms 3, 10 and 32. You may specify as many dihedral cosine terms on this line as you wish from 1 to MAXDIHTERMSPERDIHEDRAL (default 6). As in the bond, angle and ureybradley modes, *id* is a uniquely identifying number, in this case from 1 to MAXDIHPARAM (default 5000), with ID 0 referring to the special, null dihedral.

As in the case of the bond, angle and ureybradley modes, the AMBER atom set is used to associate dihedral parameters with actual dihedrals in a molecule. The final valid line associates the AMBER atom types of the four atoms in the dihedral with the dihedral parameter ID, e.g.

```
atm amb1 amb2 amb3 amb4 id
```

Because dihedrals are symmetrical, *amb1-amb2-amb3-amb4* is equivalent to *amb4-amb3-amb2-amb1*.

mode born

This mode is used to read the Generalised Born parameters that are used in the simulation. A valid line is

```
par id atype iborn scalefac
```

where atype is an AMBER/GAFF atom type, iborn is an intrinsic born radius and scalefac a scaling factor for Pairwise Descreening Approximation calculations. These parameters have been optimised to be used with the AMBER or GAFF force fields.

mode surface

This mode is used to read surface area parameters that are used in the simulation. A valid line is

```
par id atype radius surftens
```

where atype is an AMBER/GAFF atom type, radius is the radius of the atom and surftens the surface tension of this atom type, which relates the solvent accessible surface area of this atom to a non polar energy. These parameters have been optimised to be used with the AMBER or GAFF force fields.

3.8.2 Templates

Templates are used to assign the z-matrix and forcefield parameters to loaded molecules. Templates are read in using the template mode of the parameter / forcefield files (see section 3.8.1). Different types of template are used with the different types of molecules in ProtoMS;

proteins The backbone of each protein residue is assigned via a chain template. The sidechain of each residue is assigned via a residue template.

solutes Solutes are assigned via solute templates.

solvents Solvents are assigned via solvent templates.

Chain Templates

Chain templates are used to assign the z-matrix and parameters of the backbone of protein residues. The start of a new chain template is indicated by the line;

```
#
#        --  HN       O  --
#         |   |       |  |
#    res-1| --N--CA--C--|res+1
#         |       |     |
#        --       X     --
#
mode template
chain aacenter
bbatom 1  N  3  3
bbatom 2  CA 6  6
bbatom 3  C  1  1
bbatom 4  O  2  2
atom  HN 4 4  N  CA  C
zmat  HN 1.010 119.8 180.0
bond  O  C
bond  C  CA
bond  CA N
bond  HN N
angle HN N CA flex 3.0
dihedral HN N CA C flex 3.0
# Now the parameters
mode clj
par 3 N  7 -0.570 3.250 0.170 # N, sp2 N in amide
par 6 CH 6  0.200 3.800 0.080 # CA, sp3 C with 1 H
par 1 C  6  0.500 3.750 0.105 # C, carbonyl C
par 2 O  8 -0.500 2.960 0.210 # O, carbonyl O
par 4 H  1  0.370 0.000 0.000 # HN, amide hydrogen
```

Figure 3.6: The chain template for an amino acid backbone in the middle of the chain.

```
chain name
```

where *name* is the name of the chain template. This name uniquely identifies this chain template. If another chain template has been loaded with this name, then this chain template will overwrite it and a message will be output to the WARNING stream.

The chain template for an amino acid backbone is shown as an example in figure 3.6. The valid lines that comprise a chain template are;

```
bbatom id nam par0 par1
```

This line identifies which are the four bbatoms of the residue (see section 3.1.1 and figure 3.1). *id* identifies which bbatom this atom is (from 1 to 4), *nam* gives the name of the atom (maximum of four characters), and *par0* and *par1* are the CLJ parameters for this atom at $\lambda = 0.0$ and $\lambda = 1.0$, and these must refer to a valid CLJ parameter (from 0 to MAXCLJ, default 10000). Note that CLJ parameter 0 is used to assign a dummy atom. The name of the atom is the same as that given in the PDB file for the protein, and is limited to a maximum of four characters. The atom name must uniquely identify the atom within the residue, so this name must not be used elsewhere within this chain template, or in any residue templates that connect to this chain template.

```
atom nam par0 par1 bndnam angnam dihnam
```

This line identifies any extra atoms that are part of the backbone. *nam*, *par0* and *par1* have the same meanings as for the bbatom line. This is a z-matrix line (see appendix **??**), and *bndnam*, *angnam* and *dihnam* are the names of the atoms that are the reference from which the coordinates of this atom are generated (bond, angle and dihedral atoms). Note that this line does not state that there is a bond, angle or dihedral with these atoms. This line only says that these three atoms are used to construct this extra atom. Note that the atoms in a residue are built in sequence, so the bond, angle and dihedral atoms in this line must refer to atoms that were previously listed in the chain template.

```
zmat nam bndval angval dihval
```

This line provides the default values of the internal z-matrix coordinates for the atom called *nam*. *bndval*, *angval* and *dihval* are the default values of the bond length, angle size and dihedral size. This line is optional, and is only required if you either want ProtoMS to construct this atom if it is missing from the PDB file, or if you want ProtoMS to reset bond and angles to default values.

```
bond nam1 nam2
```

This line adds a bond between atoms name *nam1* and *nam2*. These two atom names must be present in the chain template. ProtoMS will not automatically add any bonds between atoms (except inter-residue bonds), so you must add all bonds that are present in the chain template. ProtoMS will use all of these explicitly added non-dummy bonds between atoms to generate all of the implicit angles and dihedrals within the backbone.

Additional arguments may be present on this line to control the type of bond that is added, e.g.

```
bond nam1 nam2 dummy
```

adds a dummy bond between atoms *nam1* and *nam2*. A dummy bond is really a non-bond, as it has no energy, and its presence forces ProtoMS to treat atoms *nam1* and *nam2* as though they were not bonded together.

You can make this bond flexible by adding the flex argument to the bond line, e.g.

```
bond nam1 nam2 flex delta
```

where *delta* is the maximum change in the bond length attempted in a Monte Carlo move in Å. Note that you can only make degrees of freedom flexible if they are used in the construction of the z-matrix, i.e. atom *nam1* must be constructed via a bond with *nam2*, or *nam2* constructed from *nam1*.

The forcefield parameters for this bond will normally be assigned via the AMBER atom types of the constituent atoms. It is possible to override this assignment by explicitly assigning bond parameters, e.g.

```
bond nam1 nam2 param par0 par1
```

where *par0* and *par1* are the bond parameter IDs for this bond at $\lambda = 0.0$ and $\lambda = 1.0$. The bond parameter IDs must refer to valid bond parameters (0 to MAXBNDPARAM, default 5000), where parameter 0 is used to refer to a null bond. You can use parameter 0 to state that two atoms are bonded, but that the energy of the bond should not be evaluated, e.g.

```
bond nam1 nam2 param 0 0
```

Angles and Urey-Bradley terms and dihedrals in the chain template are specified almost identically as for the bond line, e.g.

```
angle nam1 nam2 nam3
```

adds an angle between atoms named *nam1*-*nam2*-*nam3*,

```
ureybradley nam1 nam2 nam3
```

adds a Urey-Bradley term between atoms named *nam1*-*nam2*-*nam3*, and

```
dihedral nam1 nam2 nam3 nam4
```

adds a dihedral between atoms named *nam1*-*nam2*-*nam3*-*nam4*. `dummy` and `param` options may be added to all of these lines, and `flex` may be added to the `angle` and `dihedral` lines (where *delta* is given in units of degrees).

ProtoMS uses the bonds specified in the template to work out where all of the implicit angles and dihedrals are in the backbone. You do not need to include implicit (additional) angles or dihedrals in the template file, and you can just the template to just the flexible angles and dihedrals. However there are some cases where you would not wish an implicit angle or dihedral to be evaluated, for example the dihedral energy may only need to be evaluated via one of the dihedrals around a bond, and not via any additional dihedrals. If this is the case then you will need to add those additional dihedrals to the template and use the *dummy* keyword to specify that these are dummy dihedrals and that their energy should not be evaluated.

It is not possible to add multiple bonds between the same pair of atoms, or multiple angles to the same triplet of atoms etc. ProtoMS will only use the first definition of a bond, angle, dihedral or Urey-Bradley term and will ignore any further attempts to set them.

Figure 3.6 shows the complete chain template for an amino acid backbone in the middle of a protein chain.

```
# ALANINE
#
#      N-CA-C
#         |
#        CB
#
mode template
residue ALA
info rotate 0.5 translate 1.0
backbone first aanterm middle aacenter last aacterm single aasingle
atom  CB  7 7 CA N C
zmat  CB  1.525 111.1 -120.0
bond  CB CA
angle CB CA N flex 0.5
#parameters
mode clj
par 7 C3 6  0.000 3.910 0.160 # CB, sp3 with 3 H
```

Figure 3.7: The residue template for alanine in the OPLS united atom forcefield.

This figure shows that it can be convienient to combine the chain template with the CLJ parameters for the template into a single parameter file.

Residue Templates

Residue templates are used to assign the z-matrix and forcefield parameters for the sidechains of protein residues. The format of a residue templates is almost identical to that of a chain template. The start of a new residue template is signalled by the line;

```
residue name
```

where *name* is the name of the residue template. This name uniquely identifies the template and because residues locate templates via the residue name, the residue template name is limited to a maximum of four characters. The residue template for alanine is shown in figure 3.7 as an example. The lines that comprise a residue template are;

```
info rotate rotdel translate trandel
```

This line provides information about the residue template. The option `rotate` *rotdel* specifies that the backbone rotation move would rotate the backbone by a maximum of *rotdel* degrees. The option `translate` *trandel* specifies that the backbone translation move would translate the backbone by a maximum of *trandel* Å. Both of these options are optional, and may appear in any order on this line. If these options are not given, then the default translation and rotation values are both 0.0.

```
backbone position chain
```

This line states which chain templates are associated with this residue template for different positions of the residue within the protein, e.g.

```
# GLYCINE - this consists only of the glycine backbones
#
#     --C--CA--N--
#
mode template
residue GLY
info rotate 0.5 translate 1.0
backbone first glynterm middle glycenter last glycterm single glysingle
# glycine has no atoms, or internals!
```

Figure 3.8: The residue template for glycine in the OPLS united atom forcefield.

```
backbone first aanterm middle aacenter last aacterm single aasingle
```

states that this residue template uses the chain template called `aanterm` if this was the first residue in the protein, `aacenter` if this residue was in the middle of the protein, `aacterm` if this was the last residue in the protein, and `aasingle` if this was the only residue in the protein. You can place as many positions on this line as you wish, with possible positions being `first`, `middle`, `last` and `single`. You do not need to specify a chain template for every one of these positions, but ProtoMS will print a message to the WARNING stream if it needs a position that has not been specified.

The remaining lines in the residue template are the `atom`, `zmat`, `bond`, `angle`, `ureybradley` and `dihedral` lines, which have exactly the same meaning and formats as those in the chain template lines. Note that the names of atoms in the residue template must be different to those in any of its associated chain templates. Also note that you can (and indeed will have to!) refer to atoms that are present in the associated chain templates. In the example in figure 3.7 you can see that the only atom in the residue template is the united-atom 'CB', and that this is built from the 'CA', 'N' and 'C' atoms of its associated chain templates. This means that all of the chain templates associated with this residue template must include atoms named 'CA', 'N' and 'C'. If these atoms don't exist then ProtoMS will print many messages to the WARNING stream, and the simulation will fail.

ProtoMS will use the non-dummy bonds present in the residue template to find all of the implicit (additional) angles and dihedrals. If one of the bonds connect the sidechain to the backbone (one of the bonds should!), then the implicit angles and dihedrals between the sidechain and backbone will also be found. If you do not want the energy of these implicit angles and dihedrals to be evaluated then you need to specify them in the residue template with the `dummy` option set.

It is possible for a residue template to contain no atoms! While this may sound strange, it is necessary for residues such as glycine in united atom forcefields (see figure 3.8), or for some terminating residues (e.g. methylamine).

```
mode template   # make sure that the parameter file is being read in
                # in template mode

#                  |
#     CH3--CH2     |     CH2--CH3        Biphenyl, built as two residues,
#    /        \    |    /        \             PH1 and PH2
#  CH4        CH1-|-CH1         CH4
#    \        /    |    \        /        Note that each atom in a residue
#     CH5--CH6     |     CH6--CH5         must have a unique name but
#  PH1             |              PH2     that atoms in different residues
#                                         may have the same name
solute biphenyl
info translate 1.0 rotate 5.0

# Atoms in the first, PH1 residue
#
atom CH1 PH1  20 20  DM3 DUM  DM2 DUM  DM1 DUM  # First three atoms are built
atom CH2 PH1  20 20  CH1 PH1  DM3 DUM  DM2 DUM  # from the auto-generated
atom CH3 PH1  20 20  CH2 PH1  CH1 PH1  DM3 DUM  # dummy atoms (DM1-DM2-DM3)
atom CH4 PH1  20 20  CH3 PH1  CH2 PH1  CH1 PH1
atom CH5 PH1  20 20  CH4 PH1  CH3 PH1  CH2 PH1
atom CH6 PH1  20 20  CH5 PH1  CH4 PH1  CH3 PH1

# Atoms in the second, PH2 residue
#
atom CH1 PH2  20 20  CH1 PH1  CH2 PH1  CH3 PH1
atom CH2 PH2  20 20  CH1 PH2  CH1 PH1  CH2 PH1
atom CH3 PH2  20 20  CH2 PH2  CH1 PH2  CH1 PH1
atom CH4 PH2  20 20  CH3 PH2  CH2 PH2  CH1 PH2
atom CH5 PH2  20 20  CH4 PH2  CH3 PH2  CH2 PH2
atom CH6 PH2  20 20  CH5 PH2  CH4 PH2  CH3 PH2
# Bonds between atoms - residue PH1
bond CH1 PH1  CH2 PH1
bond CH2 PH1  CH3 PH1
bond CH3 PH1  CH4 PH1
bond CH4 PH1  CH5 PH1
bond CH5 PH1  CH6 PH1
bond CH6 PH1  CH1 PH1
# interconnecting bond
bond CH1 PH1  CH1 PH2
# bonds in residue PH2
bond CH1 PH2  CH2 PH2
bond CH2 PH2  CH3 PH2
bond CH3 PH2  CH4 PH2
bond CH4 PH2  CH5 PH2
bond CH5 PH2  CH6 PH2
bond CH6 PH2  CH1 PH2

# only one flexible dihedral - interconnecting dihedral
dihedral CH2 PH2  CH1 PH2  CH1 PH1  CH2 PH1  flex 5.0
```

Figure 3.9: The solute template for a united atom biphenyl molecule.

Solute Templates

Solute templates are used to assign the z-matrix and forcefield parameters to solute molecules. An example solute template for a united atom biphenyl is shown in figure 3.9. The format for a solute template is very similar to that of a residue template. The main difference is that while residue atoms are uniquely identified by thier atom name, solute atoms are uniquely identified by the combined atom name and residue name, e.g. the biphenyl atom `CH2 PH2` is a different atom to `CH2 PH1`.

A new solute template is started with the line;

```
solute name
```

where *name* is the uniquely identifying name of the solute template. As with the other templates, if a solute template with this name already exists, then it is overwritten by the new template. The name of the solute template can be any length up to 300 characters that can include spaces. Valid solute names thus include 'biphenyl' and 'test ligand 132B' Note that ProtoMS is insensitive to case, so it doesn't matter how you capitalise the solute name as ProtoMS will ignore it. The solute names 'biphenyl', 'BIPHENYL' and 'BiPhenyl' are all equivalent. ProtoMS will also strip the spaces before and after the solute name, and will replace multiple spaces within the name with single spaces, e.g. ' test ligand 132B ' is equivalent to 'test ligand 132B'.

The format and meaning of the valid lines in a solute template file are very similar to those of a residue and chain template. The line;

```
info rotate rotdel translate trandel
```

has exactly the same format for a solute template as it does for a residue template, and the meaning is very similar. In this case this line sets the maximum amounts that the solute molecule as a whole will be rotated and translated by, in units of Å and degrees respectively. This line is optional, and it is not present then the default maximum rotation and translation amounts are both zero. Note that translation and rotation of a solute is about the location of the first automatically added dummy atom at the center of geometry of the solute.

```
atom nam res par0 par1 bnd bndres ang angres dih dihres
```

This line has a very similar meaning to the `atom` line of the residue and chain templates. In this case, this line identifies the solute atom called *nam*, in residue named *res*, and assigns it the CLJ parameters *par0* at $\lambda = 0.0$ and *par1* at $\lambda = 1.0$. The bond, angle and dihedral z-matrix atoms that are used to build this atom are the atom named *bnd* in residue *bndres*, the atom named *ang* in residue *angres* and the atom named *dih* in residue *dihres*. These z-matrix atoms must have appeared in the solute template before this atom. Note that this line does not add a bond, angle or dihedral between any of these atoms. The atom lines only specify how to move and construct the solute, not how to evaluate its energy. Appendix **??** describes how to construct a z-matrix and gives z-matrices for common molecular motifs.

```
bond nam1 res1 nam2 res2
```

This line adds a bond between solute atoms *nam1* in residue *res1* and *nam2* in residue *res2*. You can make this bond flexible by using the `flex` keyword in the same way as described for the chain and residue templates

(as long as this bond is used in one of the `atom` z-matrix lines to construct one of the atoms). You can also use the same `dummy` keyword as the chain and residue templates to turn this into a dummy bond. As in those cases, a dummy bond is a non-bond, and has the effect of stating that the two atoms are not bonded together. The forcefield parameters for this bond are obtained via the AMBER types of the two solute atoms. However these parameters may be overridden through the use of the `param` keyword as used in the chain and residue templates, e.g.

```
bond nam1 res1 nam2 res2 param par0 par1
```

This line states that this bond uses bond parameter *par0* at $\lambda = 0.0$ and bond parameter *par1* at $\lambda = 1.0$.

The angles, Urey Bradley terms and dihedrals in the solute are specified in a very similar manner;

```
angle nam1 res1 nam2 res2 nam3 res3
```

```
ureybradley nam1 res1 nam2 res2 nam3 res3
```

```
dihedral nam1 res1 nam2 res2 nam3 res3 nam4 res4
```

The `dummy`, `flex` and `param` options may be used with these lines, with the exception of the `ureybradley` line, which cannot use the `flex` option.

ProtoMS only uses the bonds listed in the solute template to work out which atoms are bonded together. ProtoMS does not try to guess which atoms are bonded together, so you will need to add all bonds that exist in the solute to the template file to ensure that the intramolecular energy is calculated correctly. ProtoMS will use these explicitly added, non-dummy bonds to work out all of the implicit (additional) angles and dihedrals in the solute. You do not need to include any additional angles or dihedrals in the solute template as they are added automatically by ProtoMS. If you do not want the energy of an additional angle or dihedral to be evaluated then you will need to add it to the template with the `dummy` option set. This is the same behaviour as in the chain and residue templates.

Solute templates have one extra type of valid line compared to chain or residue templates. This line is used to describe how the geometry of the solute changes with λ;

```
variable nam res type val0 val1
```

nam and *res* are the name and residue of the atom that changes geometry with λ. *typ* can be either *bond*, *angle* or *dihedral* and describes whether the bond, angle or dihedral changes with λ, with *val0* giving its value at $\lambda = 0.0$ and *val1* giving its value at $\lambda = 1.0$. These variable geometry lines are very useful for free en-

ergy calculations where an atom is being 'switched off' by turning it into a dummy atom. You can use the variable geometry line to shrink the bond length to its z-matrix bonded atom, thus having the effect of pulling it within the van der waals sphere of the bonded atom. This prevents instabilities that may arise when the atom is close to being fully switched off. See sections **??** and **??** in the next chapter for examples of this type of perturbation.

Another use for variable geometry lines is to perform free energy calculations along structural coordinates, e.g. pulling two molecules apart. You can perform these sorts of calculations in ProtoMS by loading both molecules as a single solute, with no bonds between the two molecules. You could then use a variable geometry line to change the distance between the two molecules with respect to λ. See section **??** in the next chapter for an example of this sort of free energy calculation.

Yet another use of geometry variation is to calculate the energy along an internal degree of freedom, e.g. by performing a torsion drive for the purposes of generating a dihedral forcefield parameter. See NEED EXAMPLE in the next chapter for an example of this type of calculation.

While λ may be used to change the forcefield parameters of any atom of any molecule in the entire system, only solutes may have their geometry changed with respect to λ. This is because geometry variations are implemented by making two copies of the solute and using these to shadow the original, reference solute. While you will not see these shadow solutes, they will reduce the number of solutes that you can load by two for every solute of variable geometry that you load. This means that while you can load a maximum of 50 solutes, you can only load a maximum of 16 solutes that have variable geometry.

Solvent Templates

Solvent molecules are implemented as rigid molecules in ProtoMS, so they do not require a z-matrix, nor do they have any internal degrees of freedom or energy terms. Solvent templates are thus much more simple than chain, residue and solute templates as they are only used to assign the forcefield parameters of the solvent molecules. An example solvent template for TIP4P water is shown in figure 3.10.

A new solvent template is signified by the line;

```
solvent name
```

where *name* is the uniquely identifying name of the solvent template. As in the cases of the other templates, if a solvent template with this name has been previously loaded, then it is overwritten. Solvent molecules are named using the residue name column from the PDB file, so the solvent name is limited to four characters.

There are only two types of line that are valid within a solvent template. These are an `info` line, that has the

```
#
# TIP4P (T4P)
#
#      O00         dist(OH) = 0.9572 A
#    /  |  \       dist(OM) = 0.15 A
# H01 M03 H02      ang(HOH) = 104.52 deg
#

mode clj
par 2003  OW  8   0.000   3.15363   0.1550
par 2004  HW  1   0.520   0.0       0.0
par 2005  ??  0 -1.040   0.0       0.0

mode template
solvent T4P
info translate 0.15 rotate 15.0
atom O00 2003 2003
atom H01 2004 2004
atom H02 2005 2005
atom M03 2006 2006
```

Figure 3.10: The solvent template for the TIP4P solvent model. The TIP4P CLJ parameters for the preceed the solvent template itself.

same meaning as that in the solute templates, and;

```
atom nam par0 par1
```

which states that the solvent atom called *nam* has CLJ parameters *par0* at $\lambda = 0.0$ and *par1* at $\lambda = 1.0$.

The file `solvents.ff` in the `parameter` directory contains the solvent templates for a large number of standard solvents. All of the CLJ parameters used in this file range from 2001 to 2999.

3.8.3 Protein File

Proteins are loaded from protein files. The names of the protein files are specified using the `proteinN` command described in section 3.4. The protein file is just a standard PDB format file. The name of the protein contained within this file is taken from the `HEADER` line of the PDB (see figure 3.11). The protein name may contain spaces, though ProtoMS will strip any spaces before or after the name, and will collapse multiple spaces into a single space (much like it does with the solute name, as described in section 3.8.2).

ProtoMS tries to follow the PDB format when it reads in PDB lines (see http://www.rcsb.org/pdb/docs/-format/pdbguide2.2/guide2.2_frame.html). Atom names and coordinates are given on lines that start with `ATOM` or `HETATM`. As with the rest of ProtoMS, the capitalisation of these keywords is not important. Unlike the rest of ProtoMS, these lines have a strict format with respect to in which column each piece of data is recorded. Figure 3.12 describes this strict column-based format.

ProtoMS constructs the protein chain from the residue order that it reads in from the PDB file. This means that if a protein file contains residues numbered 5, 10 and 2, in that order, then ProtoMS will construct a protein chain

```
HEADER p38 kinase (pdb1a9u.ent)
ATOM      4  HN1 GLU     1      22.816 -16.131  -8.691
ATOM      5  N   GLU     1      23.121 -16.281  -9.631
ATOM      6  CA  GLU     1      21.959 -16.152 -10.557
ATOM      7  HN2 GLU     1      23.814 -15.601  -9.869
ATOM      8  HN3 GLU     1      23.512 -17.198  -9.713
ATOM      9  C   GLU     1      22.356 -16.372 -12.036
ATOM     10  CB  GLU     1      21.306 -14.762 -10.382
ATOM     11  CG  GLU     1      19.966 -14.774  -9.628
ATOM     12  CD  GLU     1      19.601 -13.428  -9.006
ATOM     13  OE1 GLU     1      20.379 -12.933  -8.165
ATOM     14  OE2 GLU     1      18.533 -12.869  -9.353
ATOM     15  O   GLU     1      23.080 -17.316 -12.372
ATOM     16  N   ARG     2      21.859 -15.490 -12.899
ATOM     17  CA  ARG     2      22.111 -15.506 -14.341
ATOM     18  C   ARG     2      21.698 -16.727 -15.143
ATOM     19  HN  ARG     2      21.272 -14.765 -12.537
ATOM     20  CB  ARG     2      23.575 -15.195 -14.658
ATOM     21  CG  ARG     2      23.741 -14.488 -16.010
ATOM     22  CD  ARG     2      23.054 -13.119 -16.007
ATOM     23  NE  ARG     2      22.388 -12.812 -17.269
ATOM     24  CZ  ARG     2      21.572 -11.779 -17.449
ATOM     25  HE  ARG     2      22.502 -13.366 -18.094
ATOM     26  NH1 ARG     2      21.316 -10.944 -16.451
ATOM     27  H11 ARG     2      20.705 -10.165 -16.595
```

Figure 3.11: The first few lines of a protein PDB file that may be read by ProtoMS. These are the first two residues of p38 kinase (PDB code 1A9U). ProtoMS will read the name of this protein as `p38 kinase (pdb1a9u.ent)`.

```
12345678901234567890123456789012345678901234567890123456789012345678901234
ATOM  NUMBR NAME RNAM RNUM    Xxxx.xxxYyyy.yyyZzzz.zzz
ATOM     27  H11 ARG     2      20.705 -10.165 -16.595

12345678901234567890123456789012345678901234567890123456789012345678901234
ATOM  NUMBR NAME RNAM RNUM    Xxxx.xxxYyyy.yyyZzzz.zzz
HETATM   24  CZ  ARG     2      21.572 -11.779 -17.449
```

Figure 3.12: The column based format of PDB `ATOM` and `HETATM` lines. Columns 1-6 contain either `ATOM` or `HETATM`. Columns 7-11 contain the id number for the atom (from 1 to 99999). Columns 13-16 contain the name of the atom (from 1 to 4 characters). Columns 18-21 contain the name of the residue that the atom is in (from 1 to 4 characters). Columns 23-26 contain the number of the residue that this atom is in (from 1 to 9999). Columns 31 to 38 contain the x-coordinate of the atom, formatted as an 8.3 float. Columns 39 to 46 contain the y-coordinate, also as an 8.3 float, and columns 47 to 54 contain the z-coordinate, also as an 8.3 float. With the exception of the coordinates, ProtoMS does not mind exactly how each field is formatted within its allocated columns, e.g. the atom name `CZ` could be in columns 13-14 or 15-16 (thus '`CZ  `' is equivalent to '`  CZ`'). ProtoMS does not use the atom or residue numbers, so these can be any value that you desire, with the constraint that all atoms in the same residue have the same residue number, and that no two residues may have the same number. When ProtoMS writes a PDB file it will wrap any atom or residue numbers that are too large back to 1.

with the sequence 5-10-2. ProtoMS will not try to be clever and numerically order your residues for you! One requirement when loading a protein PDB is that all atoms that are part of a residue are together within the PDB file. It is not possible to scatter atoms from one residue throughout the entire PDB file. In addition, all residues in the protein must have a unique residue number, and all atoms within the same residue must have unique names.

ProtoMS loads the protein and assigns residue templates based on the residue names that it finds in the PDB file. If ProtoMS cannot find a residue template that matches the residue name then it prints a message to the WARNING stream and then skips the residue. ProtoMS will use the residue and chain templates that it finds to work out which atom names should be present in the residue. If the PDB file provides an atom that matches the atom name, then ProtoMS assigns that atom from the template. If the PDB file does not provide an atom that matches the name, then if the atom name corresponds to one of the required bbatoms, then ProtoMS will print a severe message to the WARNING stream and will then skip the residue. If the missing atom is not a bbatom, then if the residue or chain templates provide `zmat` information for that atom then the coordinates for the atom are constructed automatically (and a message output to the WARNING stream). If no `zmat` information is available for this atom, then it is skipped and a severe message is output to the WARNING stream. Finally, if the PDB file provides an atom that is not part of the template, then that atom is skipped.

ProtoMS can only read a single protein chain from a PDB file. This means that you must split multi-chain PDB files into several files, and that PDBs using the 'A' or 'B' chain notation will be read incorrectly. If ProtoMS reads `TER` line, then it will print a message to the WARNING stream, and will then skip the rest of the PDB file.

ProtoMS is capable of reading a wide variety of PDB files, and of fixing many of the errors that it encounters. Despite this, I would recommend that you do not just use a PDB direct from the databank, but that you first preprocess the PDB with another software package to ensure that the PDB is correct, and that polar hydrogens and titratable residues are included correctly.

3.8.4 Solute File

Solute input files are very similar to protein input files. Solute files are standard PDB format coordinate files. The name of the solute is read from the `HEADER` line in an identical manner to the name of a protein (see figure 3.11). The solute name is used to locate the solute template, which is used to assign the z-matrix and forcefield parameters of the solute.

The solute PDB file has the same format as a standard PDB (see figure 3.12), with the requirements that all atoms belonging to a residue are together in the PDB, that each residue name is unique, and that all atom names within a residue are unique. A valid PDB solute file for biphenyl is shown in figure 3.13. This can be compared

```
header biphenyl
ATOM      1  CH1 PH1     1       0.000   0.000   0.000
ATOM      2  CH2 PH1     1       0.000   0.000   1.400
ATOM      3  CH3 PH1     1       1.212   0.000   2.100
ATOM      4  CH4 PH1     1       2.424   0.000   1.400
ATOM      5  CH5 PH1     1       2.424   0.000   0.000
ATOM      6  CH6 PH1     1       1.212   0.000  -0.700
ATOM      7  CH1 PH2     2      -1.255   0.000  -0.725
ATOM      8  CH2 PH2     2      -2.468   0.000  -0.025
ATOM      9  CH3 PH2     2      -3.680   0.000  -0.725
ATOM     10  CH4 PH2     2      -3.680   0.000  -2.125
ATOM     11  CH5 PH2     2      -2.468   0.000  -2.825
ATOM     12  CH6 PH2     2      -1.255   0.000  -2.125
```

Figure 3.13: The solute PDB file for biphenyl. This PDB file may be used with the solute template shown in figure 3.9.

to the solute template for biphenyl, shown in figure 3.9

As is the case for protein files, ProtoMS will only read a single solute from each solute PDB file, and will skip the rest of the solute PDB if it encounters a `TER` line. It is intended that a future version of ProtoMS will remove this restriction.

ProtoMS will use the solute name to find the solute template for this molecule, and will then try to locate each atom from the template within the PDB file. If the atom does not exist then ProtoMS can automatically build the missing atom as long as its `zmat` information has been provided. If ProtoMS cannot build the atom then it skips it, after writing severe messages to the WARNING stream. If the PDB contains atoms that are not listed in the template then these atoms are ignored.

3.8.5 Solvent File

Solvent input files are very similar to protein and solute input files. Solvent files are standard PDB format coordinate files (see figure 3.12). Unlike the protein and solute files, many solvent molecules may be contained within each solvent input file. The name of each solvent molecule is taken from the residue name, and it is this name that is used to locate the template for each solvent molecule. ProtoMS will then try to locate each atom from the template within the PDB file. If the atom cannot be found then ProtoMS will write a severe message to the WARNING stream and will skip that atom. If the PDB contains atoms that are not part of the template then they are skipped. Note that ProtoMS will take the coordinates of the solvent molecule from the PDB file and will make no attempt to ensure that the internal geometry of the solvent molecule is correct for the template model (e.g. that TIP4P water has an O-H bond length of 0.9572 Å). It is planned that a future version of ProtoMS will have this capability.

If multiple solvent files are loaded, then the solvents from the newer files are appended onto the list of solvents loaded from the previous file. If solvent file 1 contains 340 solvent molecules, and solvent file 2 contains 10 solvent

molecules, then the solvents from file 1 will be loaded as solvent molecules 1-340, and those from solvent file 2 will be loaded as solvent molecules 341-350.

Boundary conditions

As well as containing the coordinates of the solvent molecules, the solvent file may be used to specify the parameters needed for the boundary conditions. To do this, the solvent file must include a `HEADER` line that has one of the following formats;

```
HEADER box dimx dimy dimz
```

This states that the solvent file contains a box of solvent of dimensions *dimx* Å by *dimy* Å by *dimz* Å, with the box centered on the origin. Note that ProtoMS will not check to see if this information is correct, so you will need to ensure that that no solvent molecules lie outside of this box.

```
HEADER box ox oy oz tx ty tz
```

This states that the solvent file contains a box of solvent with the bottom-left-back corner located at coordinates (*ox*,*oy*,*oz*) Å and the top-right-front corner located at coordinates (*tx*,*ty*,*tz*) Å. Again ProtoMS will not check that this information is accurate!

```
HEADER cap ox oy oz rad k
```

This states that the solvent file contains solvent molecules restrained to be within a spherical cap of radius *rad* Å, centered at coordinates (*ox*,*oy*,*oz*) Å, using a half-harmonic force constant of *k* kcal mol^{-1} Å $^{-2}$. ProtoMS will not check to see whether or not this information is accurate.

Only one `HEADER` line may be included in each solvent file. How ProtoMS interprets these `HEADER` lines depends on which boundary conditions had been set for the simulation (see section 3.4).

1. If no boundaries had been set for the simulation, then any information in the solvent files is ignored.

2. If a solvent cap had been set for the simulation, then any information in the solvent files is ignored and the solvent cap parameters are taken from the simulation parameter.

3. If a solvent box had been set then the solvent box dimensions are initially taken from the simulation parameter. However if any of the loaded solvent files specify the solvent box size then the solvent box dimensions are increased to encompass both the initial dimensions and the solvent box dimensions.

4. If ‘solvent’ boundaries had been set for the simulation then the boundaries used will be those obtained from

the first solvent file that is loaded that contains a `HEADER` line. If none of the loaded solvent files contain a `HEADER` line then a warning is printed and no boundary conditions are used. Note that by default 'solvent' boundaries are set for all simulations.

Warnings are printed if solvent files contain conflicting boundary types (e.g. specifying a box when a spherical solvent cap is used), or if multiple solvent files supply solvent cap parameters. If multiple solvent files supply solvent box dimensions then the box is increased to the minimum size necessary to encompass all of the solvent boxes.

To make things simple, I recommend that you use one solvent file to describe your boundary conditions, and use the default option of specifying solvent boundaries via the solvent file (use `boundary solvent` in your command file, or do not supply a `boundary` value as `solvent` is the default).

ProtoMS will print out the boundary dimension to any output PDB file if that file contains solvent molecules.

3.8.6 Restart File

The restart file is used to save the coordinates of the entire system to a high precision such that they can be loaded up at a future point, or by another ProtoMS simulation. The format of the restart file is not yet fixed, so unfortunately there is the possibility that different versions of ProtoMS may not be able to read each other's restart files. This is considered a bug, and it is a development aim to stabilise the restart file format.

The restart file has deliberately been written as a human-readable text file. This means that the restart file is larger than it could be, but that it should be possible to manually edit a restart file, and understand its contents. If you wish to save space then I recommend that you compress the restart file via `bzip2` or `gzip`. While the restart file is human-readable and editable, I recommend that you do not attempt to change the restart file unless you have a good understanding of the `writerestart.F` and `readrestart.F` source files that are used by ProtoMS to read and write them.

The restart file only contains the coordinates of the entire system and the parameters needed for the boundary conditions. This file does not contain energies or energy averages, as these are output via the RESULTS stream. The restart file does not contain information about the connectivity or setup of the system as these are contained in the command file and the protein, solute, solvent and parameter files.

You can write a restart file at any point during your simulation, and you can write as many restart files as you wish. This means that you can start your simulation with a bit of equilibration, and write a restart file for the final equilibrated configuration, and then run some production. This is a strategy used by many of the examples in the next chapter.

You can read a restart file at any point during your simulation, and you can read restart files as many times as you desire during a simulation. A restart file merely resets the coordinates of the system to those saved when the restart file was written. This means that you could run multiple chunks of a simulation from the same equilibrated configuration by reading in a restart file from the equilibrated configuration before performing each chunk of production. Note that you can only read a restart file into the same system that was used to write that restart file. If you try to load an incompatible restart file then the program will print lots of warnings and will probably close down!

3.9 Conclusion

This chapter has provided a reference for ProtoMS and has described all of the commands and files that are used. To actually learn to use the program you are now advised to consult the tutorials that come with this release.

Appendix A

Free Energy Methods

It is very difficult to calculate the absolute free energy of large systems. However, it is possible to calculate the relative free energy of two different systems. This was first realised in the derivation by Zwanzig,[1]

$$\begin{aligned}
\Delta G_{A\to B} &= G_B - G_A \\
&= (-kT\ln Q_B) - (-kT\ln Q_A) \\
&= -kT\ln\left[\frac{Q_B}{Q_A}\right] \\
&= -kT\ln\left[\frac{\int \exp(-E_B(q)/kT)\,dq}{\int \exp(-E_A(q)/kT)\,dq}\right]
\end{aligned}$$

multiply by $1 = \exp(-E_A(q)/kT)\exp(E_A(q)/kT)$ gives,

$$\begin{aligned}
&= -kT\ln\left[\frac{\int \exp(-E_B(q)/kT)\times\exp(-E_A(q)/kT)\exp(E_A(q)/kT)\,dq}{\int \exp(-E_A(q)/kT)\,dq}\right] \qquad \text{(A.1)} \\
&= -kT\ln\left[\frac{\int \exp(-E_A(q)/kT)\times\exp\big(-(E_B(q)-E_A(q)/kT\big)\,dq}{\int \exp(-E_A(q)/kT)\,dq}\right] \\
&= -kT\ln\left[\int \frac{\exp(-E_A(q)/kT)}{Q_A}\times\exp(-\Delta E_{AB}(q)/kT)\,dq\right] \\
&= -kT\ln\left[\int p_A(q)\times\exp(-\Delta E_{AB}(q)/kT)\,dq\right], \\
&= -kT\ln\Big\langle \exp(-\Delta E_{AB}(q)/kT)\Big\rangle_A
\end{aligned}$$

where $P_A(q)$ is the Boltzmann probability of configuration q in the ensemble of state A, ΔE_{AB} is the difference in energy between system A and system B and $< ... >_A$ represents the average over all of the ensemble of configurations of system A. This derivation lies at the heart of all of the multi-value free energy methods implemented in ProtoMS.

A.1 Multi-value Free Energy Methods

ProtoMS implements three multi-value free energy methods; Free Energy Perturbation (FEP), Finite Difference Thermodynamic Integration (FDTI) and Replica Exchange Thermodynamic Integration (RETI). Multi-value free energy methods are so called because the energy for multiple values of λ are evaluated at each step of the simulation.

A.1.1 Free Energy Perturbation

Free Energy Perturbation[1] (FEP) is a rigorous free energy method that has been used to calculate binding free energies in many successful studies, i.e. calculating the specificities of various ligands for the COX-1 or COX-2 enzymes,[2] investigating the enantioselective binding of peptide based ligands to a small host,[3] the binding of ligands to SH2,[4,5] or FK506 binding protein,[6] and the binding of alkali metal cations to spherands.[7] FEP calculates the free energy change along the λ-coordinate through direct use of the Zwanzig equation (equation A.1). To ensure that this equation converges, the λ-coordinate is split into a series of windows (figure A.1). The width between each window must be sufficiently small to ensure good overlap between the reference and perturbed states.[8] An MC or MD simulation is then run within each window, and the Zwanzig equation applied to calculate the free energy change between each window and its neighbour. The relative free energy along the λ-coordinate can then be achieved by summing each of the individual free energy differences between each window (figure A.1). The free energy differencses could be calculated between each window and the next neighbour. Summing these values would yield the free energy change for the forwards perturbation from state A to state B. Conversely, the free energy differences could be calculated between each window and its previous neighbour. Summing these would yield the free energy change for the backwards perturbation from state B to state A. If the calculation has converged, then the forwards and backwards free energies should be equal. Any difference between them is known as hysteresis, and examination of where hysteresis occurs can be used to position better the λ-windows for any subsequent calculations. To enable this analysis, FEP simulations typically calculate both the forwards and backwards perturbations, using a technique known as double-wide sampling.[9] Using this technique, a free energy

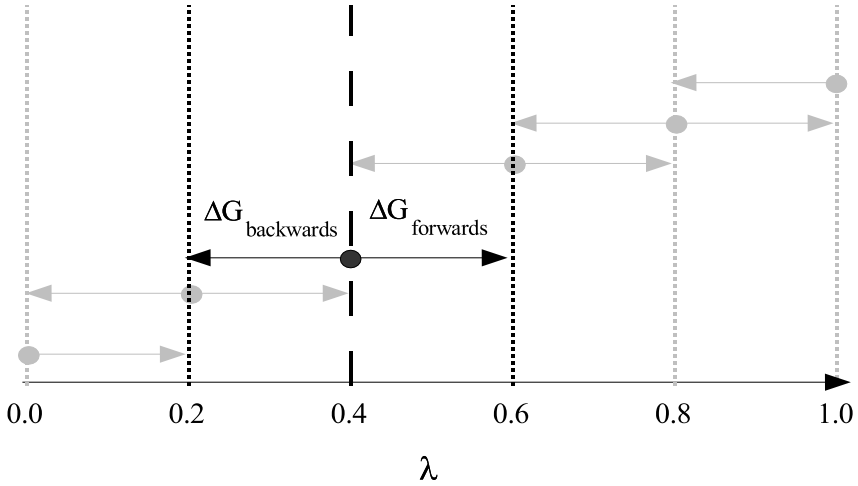


Figure A.1: The use of windows in FEP simulations. In this figure, the λ-coordinate is split up into 6 windows, spaced evenly every 0.2 λ-units. A simulation (represented by a filled circle) is run in each window. The $\lambda = 0.4$ window is highlighted. The forwards free energy to the next window, and backwards free energy to the previous window are calculated during the simulation. The sum of all the forwards free energies yields the forwards estimate of the relative free energy, while the negative of the sum of all of the backwards free energies yields the backwards estimate. Obviously, both estimates should be the same.

simulation is performed at each λ-window, and the free energy difference simultaneously estimated between the next and previous window.

FEP accumulates the exponential of the difference in energy between neighbouring windows. If the λ-windows are well positioned, then these differences in energy will be well-behaved, and their fluctuations will be small. However, the initial positioning of the windows is difficult, as there is no a priori knowledge of the shape of the potential of mean force (PMF) along λ. More windows are needed in regions where the PMF changes rapidly, while fewer are needed in regions where the PMF is flat. The technique of dynamically modified windows[10] tries to alleviate this problem by using the free energy change in one window to estimate the optimum width of the next window. It achieves this by fitting a straight line through previously calculated free energies to estimate the current gradient of the PMF. The value of this gradient can then be used to decide where to place the next λ-value. A problem with this method is that it was developed at a time when computer resources required that each λ-window were run serially, and thus the gradient information from the previous λ-windows was readily available. However, the advent of cheap, yet powerful Beowulf clusters means that all λ-windows can now be run simultaneously in parallel, and thus the spacing between all λ-windows must be determined before the simulation starts.

A.1.2 Thermodynamic Integration

Thermodynamic Integration (TI) is another rigorous free energy method with a significant history of successful applications, e.g. the calculation of the relative binding free energies of ligands to p38,[11] the estrogen receptor ligand-binding domain[12] or acetylcholinesterase,[13] the relative hydration free energy of n-alkanes,[14] the binding of ions to a calix[4]pyrrole derivative,[15] and investigating the interactions between amino acid residues in the binding site of trypsin.[16] While FEP directly uses the Zwanzig equation to calculate the difference in free energy along the λ-coordinate, TI takes a different approach. The method still looks at discrete λ-values along the coordinate, and generates an MC or MD trajectory at each λ-value. However, instead of calculating the difference in energy between neighbouring λ-values, it calculates the rate of change of free energy, with respect to λ at each point. TI thus avoids the problems of low overlap experienced in FEP, as this free energy gradient, $(\frac{\partial G}{\partial \lambda})_\lambda$, is a property of the system at each value of λ only. Once all of these free energy gradients are obtained, they may be integrated to yield the relative free energy along the λ-coordinate.

$$G_{\lambda=1} - G_{\lambda=0} = \int_0^1 \left(\frac{\partial G}{\partial \lambda}\right)_\lambda d\lambda \tag{A.2}$$

This integral can be evaluated numerically, e.g. via the trapezium rule.[11] The free energy gradients themselves may be obtained analytically or numerically. The analytical route uses a modified forcefield to calculate the gradient of each forcefield term directly with respect to λ. The ensemble average of the gradient of forcefield, $\left\langle \frac{\partial E}{\partial \lambda} \right\rangle_\lambda$, is equal to the free energy gradient.

$$\int_0^1 \left(\frac{\partial G}{\partial \lambda}\right)_\lambda d\lambda = \int_0^1 \left\langle \frac{\partial E}{\partial \lambda} \right\rangle_\lambda d\lambda \tag{A.3}$$

The numerical route approximates the gradient, $(\frac{\partial G}{\partial \lambda})_\lambda$, via the finite difference, $(\frac{\Delta G}{\Delta \lambda})_\lambda$. This free energy difference can be calculated via the Zwanzig equation, with the reference state at λ, and the perturbed state at $\lambda + \Delta\lambda$. This would give a forwards estimate of the free energy gradient. A perturbed state of $\lambda - \Delta\lambda$ yields the backwards estimate. These two estimates should of course be equal if Δλ were sufficiently small, and the trajectory ran until the Zwanzig equation had converged. This method is normally referred to as Finite Difference Thermodynamic Integration[17] (FDTI), and again, there is a significant body of literature that demonstrates its successful application. These include its application to the relative binding free energy of thrombin inhibitors[18,19,20] and ligands to DHFR.[21] Most workers who use FDTI position their windows, and integrate the results, through the use of

a Gaussian quadrature.[22] This is a technique that was developed by Gauss in the early nineteenth century to integrate definite functions. It works by recasting the integral of the function into the class of integrals known as 'polynomials times a known weighting function'. Given a function, $f(x)$, the method allows the optimal choice of the points along x to evaluate the function, and the optimal weights to give each of those points. This can be achieved by multiplying $f(x)$ by a known weighting function, $W(x)$. If $f(x)$ is a polynomial, then, given an integer number of evaluation points, N, it is possible to find a set of weights, w_j, and abscissas, x_j, such that the approximation,

$$\int_a^b W(x)f(x)dx \approx \sum_{j=1}^{N} w_j f(x_j), \tag{A.4}$$

is exact. The values of w_j and x_j depend only on the choice of the weighting function, and the number of evaluation points.[22] Most workers use Gauss-Legendre quadrature, which uses a weighting function of $W(x) = 1$. The values of x_j and w_j are obtained from a lookup table, and are not in any way based on the shape of the PMF along λ. It is our opinion that this method of integration is not best suited for application to free energy calculations, as it was designed to integrate definite functions, and only has high accuracy for functions that are well approximated by polynomials. The gradients obtained along λ will have associated errors, and the underlying PMF could contain sharp peaks or troughs. The positioning of the λ-windows should thus reflect the shape of the PMF, with more λ-windows in places where there are large changes in the free energy gradient. All gradients should also be weighted equally, so as to avoid the possibility of points with large errors being highly weighted, while reducing the contribution from points with low errors. In addition, the use of Gaussian quadrature has, in our opinion, led to the use of far too few λ-windows along the perturbation. Previous studies[18,19,20] have used as few as 6 λ-windows, while another study[21] stated that increasing the number of λ-windows from 6 to 8 reduced the quality of the results. Such a reduction in quality would have been the result of poor positioning of λ windows, not an increase in their number. Our application of FDTI will thus use the same integration methods that have been used successfully in standard TI simulations,[11] namely trapezium rule integration over many, closely spaced λ-windows. Using this scheme, FDTI is very similar to FEP, and can be run with the same system conditions and reference states. In the case of FEP, the perturbed states are the neighbouring windows, while in the case of FDTI, the perturbed states are $\Delta\lambda$ above and below each window. FDTI and FEP become identical in the limit of the window width becoming equal to $\Delta\lambda$.

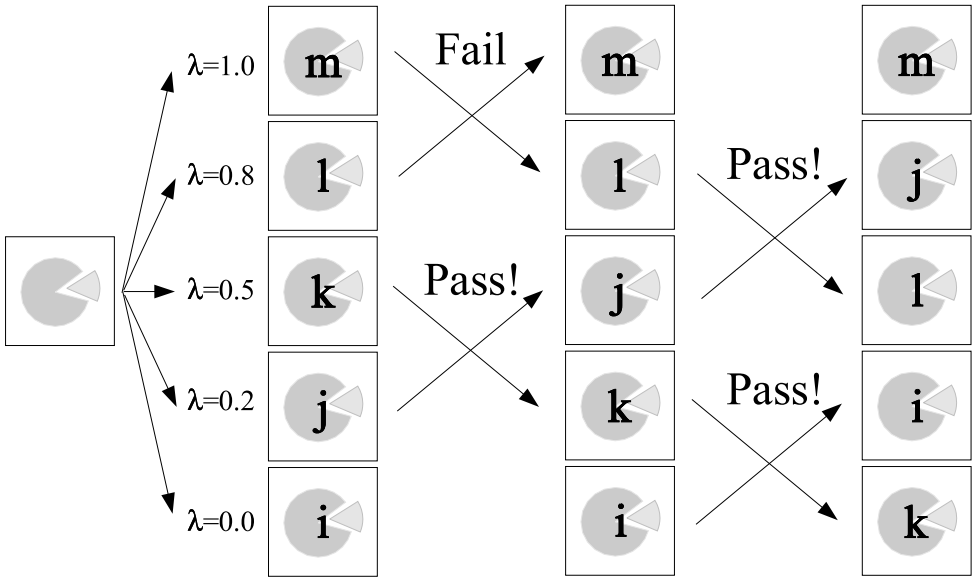


Figure A.2: The Replica-Exchange Thermodynamic Integration (RETI) algorithm.

A.1.3 Replica Exchange Thermodynamic Integration

Replica Exchange Thermodynamic Integration (RETI)[23,24] is a very recently developed free energy method that combine FDTI with replica exchange. The use of the λ-coordinate to scale the forcefield leads to the system having a different Hamiltonian at each value of λ. This means that Hamiltonian Replica-Exchange may be used within a normal FDTI simulation to periodically test and swap the coordinates of neighboring λ-values (figure A.2). The FDTI simulation may be conducted in the standard manner, with a replica of the system at each value of λ. Unlike standard FDTI, there is the addition of a periodic λ-swap move, whereby neighboring λ-replicas are tested according to the Hamiltonian Replica-Exchange test. This test is the same for the NVT and NPT ensembles. For replica i at $\lambda = A$ and energy $E_A(i)$, and replica j at $\lambda = B$ and energy $E_B(j)$, the test is given by,[25]

$$\exp\left[\frac{1}{kT}\left(E_B(j) - E_B(i) - E_A(j) + E_A(i)\right)\right] \geq rand(0,1). \tag{A.5}$$

If this test is passed, then the coordinates of the pair of λ-values are swapped. The derivation of this test[25] is similar to that used for Parallel Tempering.[26,27] It is useful to note that this test is just the product of two normal Metropolis tests; one going from configuration i to j at $\lambda = A$, and the other going from j to i at $\lambda = B$. The Replica-Exchange test is the product of these tests as both moves must occur simultaneously.

This sampling-test iteration is continued until convergence. At this point, the set of trajectories at each value of λ form a correct ensemble distribution for that value of λ, and thus the free energy average is formed correctly. The advantages of this method are that it is almost trivial to implement, with the addition of an inexpensive λ-swap move and test, and the additional book-keeping needed to keep track of which trajectories are at what λ-values. In addition, this scheme allows each trajectory to move freely across λ, as in the case of AdUmWHAM, but without the problems of Hamiltonian lag, since an umbrella potential is not involved. Also, the probability of accepting a λ-swap move will be based on the amount of overlap between neighboring λ-values. Since the FDTI integration requires good overlap to converge well, it is expected that the acceptance ratio of the λ-swap moves should be high.

ProtoMS provides support for RETI by providing a RETI simulation chunk that calculates the difference in energy of a configuration at neighbouring λ-values, and outputs this difference to the RETI stream for easy input to a script.

The RETI λ-swap acceptance test is derived from the general replica exchange test, which will now be derived. Let us imagine three different systems, A, B and C. Each system will be in a particular configuration, e.g. system A may adopt configuration i, system B may be in configuration j, and system C make be in configuration k. The probability, $p_A(i)$, that system A is in configuration i, in the NVT ensemble, is given by the Boltzmann equation,

$$p_A(i) = \frac{\exp^{-\beta_A H_A(i)}}{Q_A}, \tag{A.6}$$

where H_A is the Hamiltonian for the system A, β_A is $1/k_B T_A$, where T_A is the temperature of system A, and Q_A is the canonical partition function for the Hamiltonian. Similar equations exist to describe the probability of configuration j in system B, $p_B(j)$, and the probability of configuration k in system C, $p_C(k)$.

Now let us consider this collection of three systems as a whole. We have thus formed a "super-collection" or "super-ensemble" over these three systems. The probability that system A is in configuration i, *and* system B is in configuration j, *and* system C is in configuration k is given by the product of their three individual probabilities.

The probability of this set, $P(X)$, is thus given by,

$$\begin{aligned} P(X) &= p_A(i)p_B(j)p_C(k) \\ &= \frac{\exp^{-\beta_A H_A(i)} \exp^{-\beta_B H_B(j)} \exp^{-\beta_C H_C(k)}}{Q_A Q_B Q_C}. \end{aligned} \tag{A.7}$$

In this equation, X is a vector representing the super-configuration set (i, j, k). It is possible that the super-ensemble can exist in configuration set Y, representing (j, i, k), where systems A and B have swapped configurations. Chapter B will show that a Monte Carlo test could be performed on the move from X to Y, based on the relative probabilities of X and Y,

$$\begin{aligned} \frac{\rho_Y}{\rho_X} &= \frac{P(Y)}{P(X)} \\ &= \frac{\exp^{-\beta_A H_A(j)} \exp^{-\beta_B H_B(i)} \exp^{-\beta_C H_C(k)}}{Q_A Q_B Q_C} \\ &\quad \times \frac{Q_A Q_B Q_C}{\exp^{-\beta_A H_A(i)} \exp^{-\beta_B H_B(j)} \exp^{-\beta_C H_C(k)}} \\ &= \frac{\exp^{-\beta_A H_A(j)} \exp^{-\beta_B H_B(i)} \exp^{-\beta_C H_C(k)}}{\exp^{-\beta_A H_A(i)} \exp^{-\beta_B H_B(j)} \exp^{-\beta_C H_C(k)}} \\ &= \frac{\exp^{-\beta_A H_A(j)} \exp^{-\beta_B H_B(i)}}{\exp^{-\beta_A H_A(i)} \exp^{-\beta_B H_B(j)}} \\ &= \exp\left[-\beta_A H_A(j) - \beta_B H_B(i) + \beta_A H_A(i) + \beta_B H_B(j)\right] \\ &= \exp\left[\beta_B \big(H_B(j) - H_B(i)\big) - \beta_A \big(H_A(j) - H_A(i)\big)\right] \\ &= \exp(\Delta) \end{aligned} \tag{A.8}$$

$$\text{where } \Delta = \beta_B \big(H_B(j) - H_B(i)\big) - \beta_A \big(H_A(j) - H_A(i)\big).$$

Note how the partition functions for each system cancel out, as do the probabilities of system C. Indeed, this derivation demonstrates that *only* the probabilities of the systems exchanging configurations need to be included. This equation is thus valid for super-collections of *any number of systems*. Also, in this case, Δ was calculated for systems in the NVT canonical ensemble. It is possible to use a similar derivation to obtain Δ for many other ensembles. For example, the derivation of Δ for the NPT ensemble, where the volume, V, of the replicas must be

taken into account, is,

$$
\begin{aligned}
p_A(i) &= \frac{\exp^{-\beta_A(H_A(i)+P_AV_i)+N\ln V_A}}{Q_A} \\
P(X) &= p_A(i)p_B(j)p_C(k) \text{ and } P(Y) = p_A(j)p_B(i)p_C(k) \\
\frac{P(Y)}{P(X)} &= \frac{\exp^{-\beta_A(H_A(j)+P_AV_j)}\exp^{-\beta_B(H_B(i)+P_BV_i)}}{\exp^{-\beta_A(H_A(i)+P_AV_i)}\exp^{-\beta_B(H_B(j)+P_BV_j)}} \\
&= \exp\left[\beta_B(H_B(j)-H_B(i)+P_B(V_j-V_i))-\beta_A(H_A(j)-H_A(i)+P_A(V_j-V_i))\right] \\
&= \exp(\Delta) \\
\text{where } \Delta &= \beta_B\big(H_B(j)-H_B(i)+P_B(V_j-V_i)\big)-\beta_A\big(H_A(j)-H_A(i)+P_A(V_j-V_i)\big).
\end{aligned}
\tag{A.9}
$$

These values of Δ are correct for highly general replica exchange simulations, where the Hamiltonians, temperatures and pressure could vary between replicas. In practise, some of these parameters are kept the same, e.g. parallel tempering keeps the pressure and Hamiltonian the same between replicas. Substituting $H = H_A = H_B$ and $P = P_A = P_B$ into these values of Δ gives the NVT and NPT parallel tempering tests,

$$
\begin{aligned}
\text{NVT, } \Delta &= \beta_B\big(H(j)-H(i)\big)-\beta_A\big(H(j)-H(i)\big) \\
&= \big(\beta_B-\beta_A\big)\big(H(j)-H(i)\big) \\
\text{NPT, } \Delta &= \beta_B\big(H(j)-H(i)+P(V_j-V_i)\big)-\beta_A\big(H(j)-H(i)+P(V_j-V_i)\big) \\
&= \big(\beta_B-\beta_A\big)\big(H(j)-H(i)+P\big(V_j-V_i\big)\big).
\end{aligned}
\tag{A.10}
$$

Hamiltonian replica exchange, which forms the basis of RETI, only changes the Hamiltonians between replicas. Thus substituting $\beta = \beta_A = \beta_B$ and $P = P_A = P_B$ into the value of Δ yields,

$$
\begin{aligned}
\Delta &= \beta\big(H_B(j)-H_B(i)+P(V_j-V_i)\big)-\beta\big(H_A(j)-H_A(i)+P(V_j-V_i)\big) \\
&= \beta\big(H_B(j)-H_B(i)-H_A(j)+H_A(i)\big)+\beta P\big(V_j-V_i-V_j+V_i\big) \\
&= \beta\big(H_B(j)-H_B(i)-H_A(j)+H_A(i)\big).
\end{aligned}
\tag{A.11}
$$

The volumes are seen to drop out of this test, so both the NVT and NPT tests are equivalent. It is useful to note that this Hamiltonian replica exchange test is just the product of two normal Metropolis Monte Carlo tests; one checking if the change in configuration of j to i is valid for forcefield B, and one checking if the change in configuration of i to j is valid for forcefield A. Hamiltonian replica exchange uses the product of these two tests

as both moves happen simultaneously.

A.2 Single-value Free Energy Methods

ProtoMS includes support for three single-value free energy methods; Slow Growth, Fast Growth and Adaptive Umbrella WHAM (AdUmWHAM). Single-value methods are so called because the energy of only a single value of λ are evaluated at each step of the simulation, but the value of λ is able to change throughout the simulation.

A.2.1 Slow and Fast Growth

One of the most recently developed single-value methods is the so-called 'Fast Growth' method.[28,29] This is an evolution of the Slow Growth method, which is the subject of a recent review.[30] The slow growth method estimates the free energy change between two systems within a single simulation. It achieves this by slowly increasing the value of λ by a constant amount, $\delta\lambda$, at each simulation step, such that at the start of the simulation, $\lambda = \lambda_0$, and by the end of the simulation, $\lambda = \lambda_1$. If the simulation consists of M steps, then $\delta\lambda$ is given by,[30]

$$\delta\lambda = \frac{\lambda_1 - \lambda_0}{M}. \tag{A.12}$$

The system is constantly being perturbed at every step of the simulation. This perturbation requires an amount of work. The work required to perform the entire perturbation, W, is formed as a sum over all of the simulation steps,[30]

$$W = \sum_{i=1}^{M} \delta\lambda \left(\frac{\partial E}{\partial \lambda}\right)_{\lambda=\lambda_0+i\delta\lambda}. \tag{A.13}$$

If $\delta\lambda$ were infinitesimally small, then this perturbation would occur infinitely slowly. This would mean that the system would stay in thermodynamic equilibrium throughout the mutation, and the perturbation would occur reversibly. If this were the case, then the work required to perform this change would be equal to the free energy associated with the change, i.e. $W = \Delta G$. However, if the change occurred in a finite time, then the response of the system would lag behind the perturbation,[31] and the simulation would move out of equilibrium. The resulting change would not be reversible, and some of the work would be dissipated. The amount of work required would be larger than the free energy change,[28] giving the slow growth inequality,

$$W \geq \Delta G. \tag{A.14}$$

The problem with the slow growth method is that the system is constantly being moved out of equilibrium. This leads to the inequality in equation A.14. Recently, Jarzynski examined this inequality, and derived a remarkable equality,[28,29] now known as the Jarzynski identity.[32] Jarzynski realised that the problem with the slow growth method was that the individual system moved away from equilibrium in an unpredictable manner.[33] It is quite possible that there exists no general formula that describes the nonequilibrium distribution of the system at the end of the perturbation.[33] Jarzynski then showed that if an ensemble of systems were perturbed away from equilibrium, then the behaviour of the ensemble as a whole was predictable. Thus the ensemble of non-equilibrium statistics could be related to an average over the equilibrated ensemble. Using these ideas, the Jarzynski identity relates the change in free energy of a perturbation, to the average of the work calculated for a slow growth simulation for each member of the original equilibrated ensemble,[28]

$$\overline{\exp(-W/kT)} = \exp(-\Delta G/kT). \tag{A.15}$$

The overbar in this equation denotes an average over an ensemble of slow growth simulations. The beauty of this equality is that it is independent of the speed of the perturbation.[28]

To use this equality, an equilibrated ensemble of structures at $\lambda = \lambda_0$ must be generated. A slow growth simulation should be performed for each member of this ensemble, although it can be performed with a fewer number of steps, and thus a larger $\delta\lambda$ than normal slow growth. The work necessary for each simulation should be calculated, and the average of the exponential obtained. This average will then equal $\exp(-\Delta G/kT)$. Because the rate of change of λ is higher than for slow growth, this method is referred to as fast growth.[34] This method has been tested on the calculation of the excess chemical potential of a Lennard Jones fluid,[34] the potential of mean force between a pair of methane molecules in water,[32] and the charging of a sodium ion in water.[35] These tests demonstrated that similar results were obtained via the fast growth method compared with other free energy methods, using comparable amounts of processor time. This was despite the applications only averaging the results from a small subset of starting points from the initial ensemble (between 10 and 3334). The main benefit of the method appears to be its huge potential for coarse level parallelisation over a very large Beowulf cluster. Once the initial generation of the equilibrated ensemble is complete, each fast growth simulation could be performed in parallel on independent nodes.[32,34] The drawback of the method is that the average will only converge reliably if the fluctuations in the work are not too large.[28] This reasons for this are very similar to those used for the Zwanzig equation, which also collects the ensemble average of an exponential. In practice, this means that applications of this method also need to split up the λ-coordinate into a series of windows, and apply fast growth between

neighboring windows.[32]

A.2.2 Adaptive Umbrella WHAM

Methods have been presented that treat λ as a simulation parameter. Through special treatment of λ, these methods are able to integrate the free energy along the λ-coordinate, and in so doing, produce the potential of mean force (PMF) across λ.[36] A totally different approach becomes apparent when it is realised that λ is just another coordinate of the system. λ does not have to be treated specially, and thus it is possible to make dynamic changes in λ throughout a simulation. The calculation of the free energy along λ then becomes equivalent to the calculation of the PMF along a normal structural coordinate, for which many methods have been derived. One such method is Adaptive Umbrella WHAM (AdUmWHAM),[37,38] a method that combines adaptive umbrella sampling[39,40] with the Weighted Histogram Analysis Method (WHAM).[41]

AdUmWHAM is typically used to derive the potential of mean force along structural coordinates, e.g. for dihedral angles in a small peptide.[37] AdUmWHAM can be applied to perturbations by realising that λ can be treated as a dynamic coordinate, and that it is possible to make moves in λ throughout a trajectory. This realisation was first made in a precursor to AdUmWHAM, λ-dynamics.[42]

λ-dynamics

λ-dynamics is another rigorous free energy method that was designed to achieve enhanced sampling of ligand configurations and orientations within a binding free energy calculation.[42,43,44] The method treats λ as a dynamic coordinate, and allows motion along the λ-coordinate during normal configurational sampling. In this way, the ligands are dynamically morphing between each other during a single trajectory. This has the advantage that the ligands spend most of their time between $\lambda = 0.0$ and $\lambda = 1.0$, and thus the forcefield is much softer for the perturbing atoms. This should enhance the many configurational changes that are necessary to move through the λ-coordinate.[42] A potential disadvantage of the method is that λ could be changed too rapidly for the rest of the system to respond. The configurational sampling will thus lag behind the λ-sampling. This Hamiltonian lag is exactly the problem that is addressed by the Jarzynski equality in section A.2.1. If the λ-sampling is too rapid, then the system will move out of equilibrium, and the change in free energy will contain systematic error. Unfortunately, a priori knowledge of the system's relaxation time is not possible, so the λ-sampling must be performed as slowly as possible within the constraints of the simulation.

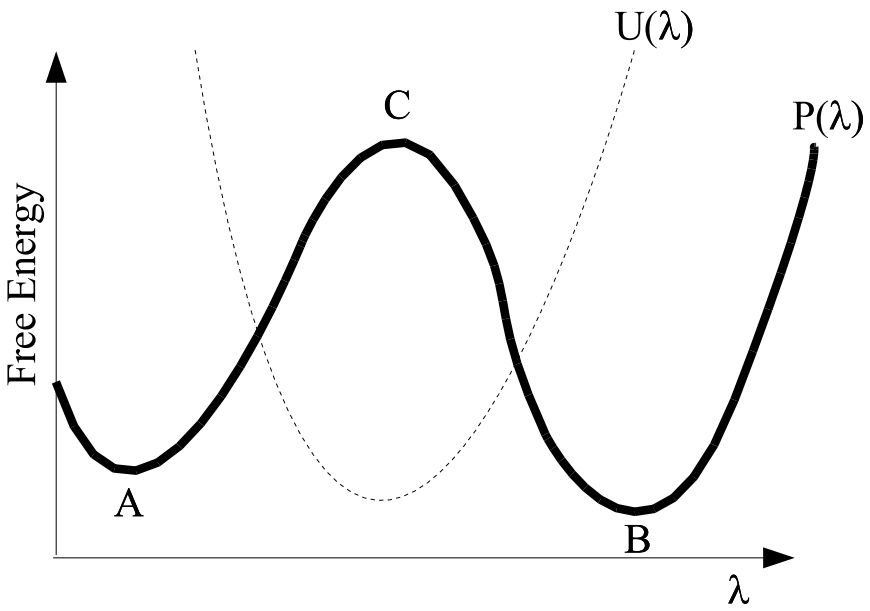


Figure A.3: A hypothetical umbrella potential which could be used to enhance sampling of the reaction coordinate, λ.

λ-dynamics uses a variable λ-coordinate to calculate the free energy. To ensure that the entire λ-coordinate is sampled fully, the motion along λ must be encouraged through the use of an umbrella potential.

Umbrella Sampling

Umbrella sampling was first developed in the late 1970s, and originally applied to a small LJ fluid test system.[45] The aim of the method is to direct the sampling along a reaction coordinate to unfavourable regions. This is achieved through biasing the simulation through use of an umbrella potential. The umbrella is implemented as an additional term to the forcefield of the system, and acts to penalise or encourage particular configurations. This has the effect of making the system sample from a biased distribution. For example, figure A.3 shows the potential of mean force, $P(\lambda)$, for a reaction coordinate, λ, for a hypothetical system. The PMF has two important minima, *A* and *B*, separated by a transition state, *C*. An umbrella potential, $U(\lambda)$, can be added to this system. This term is added to the forcefield, and has the effect of encouraging sampling where $U(\lambda)$ is low, and discouraging sampling where $U(\lambda)$ is high. In this case, the umbrella potential will encourage sampling of the transition state, thus allowing conformations *A* and *B* to interconvert freely.

The umbrella potential forces sampling from a biased probability distribution. If, in the general case, the

reaction coordinate, λ, is a function of the system coordinates, r, then the biased probability, P_b, for the particular point $\lambda = \lambda_0$ is given by,[40]

$$P_b(\lambda_0) = \frac{\int \exp\left[-\beta\big(E(r)+U(\lambda)\big)\right]\delta(\lambda-\lambda_0)dr}{\int \exp\left[-\beta\big(E(r)+U(\lambda)\big)\right]dr}, \tag{A.16}$$

where $\beta = 1/kT$, $E(r)$ is the normal energy of the system for coordinates r, $U(\lambda)$ is the value of the umbrella potential, and δ is the Dirac δ function, which is equal to one for $\delta(0)$, and zero for all other values. The use of δ ensures that only configurations of r that map to $\lambda = \lambda_0$ contribute to the probability at $P_b(\lambda_0)$. Since the form of $U(\lambda)$ is known, it is possible to re-weight this biased probability distribution to return a Boltzmann weighted

distribution along λ.[40] Looking at the specific case of the point $\lambda = \lambda_0$,

$$P_b(\lambda_0) = \frac{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]\delta(\lambda-\lambda_0)dr}{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]dr}$$

Multiplying both sides by $\exp\left(\beta U(\lambda_0)\right)$,

$$P_b(\lambda_0)\exp\left(\beta U(\lambda_0)\right) = \frac{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]\delta(\lambda-\lambda_0)dr \times \exp\left(\beta U(\lambda_0)\right)}{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]dr}$$

Since $\exp\left(\beta U(\lambda_0)\right)$ is a constant, it can be moved inside the integral,

$$P_b(\lambda_0)\exp\left(\beta U(\lambda_0)\right) = \frac{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]\delta(\lambda-\lambda_0)\exp\left(\beta U(\lambda_0)\right)dr}{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]dr}$$

The δ function has eliminated all points where $\lambda \neq \lambda_0$. $U(\lambda_0)$ now cancels,

$$P_b(\lambda_0)\exp\left(\beta U(\lambda_0)\right) = \frac{\int \exp\left[-\beta E(r)\right]\delta(\lambda-\lambda_0)dr}{\int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]dr}$$

$$\text{using } \; Q_{bias} = \int \exp\left[-\beta\left(E(r)+U(\lambda)\right)\right]dr \tag{A.17}$$

$$\text{and } \; Q_{Boltz} = \int \exp\left[-\beta E(r)\right]dr \text{ gives}$$

$$P_b(\lambda_0)\exp\left(\beta U(\lambda_0)\right) \times \frac{Q_{bias}}{Q_{Boltz}} = \frac{\int \exp\left[-\beta E(r)\right]\delta(\lambda-\lambda_0)dr}{Q_{bias}} \times \frac{Q_{bias}}{Q_{Boltz}}$$

$$P_b(\lambda_0)\exp\left(\beta U(\lambda_0)\right) \times \frac{Q_{bias}}{Q_{Boltz}} = \frac{\int \exp\left[-\beta E(r)\right]\delta(\lambda-\lambda_0)dr}{Q_{Boltz}}$$

The biased probability for $\lambda = \lambda_0$ is thus related to its

Boltzmann probability, P_{Boltz}

$$P_b(\lambda_0)\exp\left(\beta U(\lambda_0)\right) \propto P_{Boltz}(\lambda_0).$$

In the general case for any λ,

$$P_{Boltz}(\lambda) \propto P_b(\lambda)\exp\left(\beta U(\lambda)\right).$$

The unbiased probability distribution can be used to calculate the potential of mean force along λ,[36]

$$\begin{aligned} G(\lambda) &= -\frac{1}{\beta}\ln\left[P_{Boltz}(\lambda)\right] \\ &= -\frac{1}{\beta}\ln\left[P_b\exp\left(\beta U(\lambda)\right)\times\frac{Q_{bias}}{Q_{Boltz}}\right] \\ &= -\frac{1}{\beta}\ln\left[P_b\exp\left(\beta U(\lambda)\right)\right]-\frac{1}{\beta}\ln\left[\frac{Q_{bias}}{Q_{Boltz}}\right] \\ &= -\frac{1}{\beta}\ln\left[P_b\exp\left(\beta U(\lambda)\right)\right]+C. \end{aligned} \tag{A.18}$$

The constant, C, is undetermined, though its only effect is to shift the entire PMF up or down in free energy. The value of C does not affect the shape of the PMF nor the values of any relative free energies.

The form of the umbrella potential is not a priori known, so some workers use many sequential umbrella potentials, e.g. harmonic potentials, to encourage the sampling to scan successive windows along the reaction coordinate.[46] The sampling within each window can be re-weighted, and as long as the windows are overlapping, the resulting probabilities may be combined to form the PMF along the entire reaction coordinate. The combination of each of the small pieces of PMF is based on changing the values of C from equation A.18, such that the overlap between neighbouring windows is maximised. The optimal way to accomplish this is through the use of the Weighted Histogram Analysis Method.[41]

The Weighted Histogram Analysis Method

The Weighted Histogram Analysis Method[41] (WHAM) represents the optimal method of combining the statistics of multiple umbrella simulations into a single, self-consistent PMF. It achieves this by maximising the overlap in the PMF by weighting the statistics for each of the individual umbrella simulations.

The method works by dividing the reaction coordinate into a series of bins. The number of times that the sampling falls within each bin during simulation j, $n_j(i)$, is collected for each of the umbrella simulations. The value of the umbrella, for simulation j, at the centre of each bin i, $U_j(i)$ is also collected. The complete, unbiased probability for each histogram bin, i, across the reaction coordinate, $P_0(i)$, is then estimated via the self-consistent

solution to the WHAM equations,[47]

$$
\begin{aligned}
P_0(i) &= \kappa(i) \times \sum_j n_j(i) \\
\kappa(i) &= \frac{1}{\sum_j N_j f_j c_j(i)} \\
\text{where } \; f_j &= \frac{1}{\sum_i c_j(i) P_0(i)} \\
c_j(i) &= \exp\left(-U_j(i)/kT\right) \\
N_j &= \sum_i n_j(i).
\end{aligned}
\tag{A.19}
$$

A solution to these equations is obtained via an iteration. An initial estimate of $P_0(i)$ is made for each bin along the reaction coordinate. This estimate is used to calculate the weighting factor, f_j, for each simulation, which can then be used to estimate the unbiasing factor, $\kappa(i)$, for each bin in the histogram. This is then placed in the first equation to unbias the collected statistics along the reaction coordinate, and return a new estimate of $P_0(i)$. This iteration is repeated until the differences between the estimates of $P_0(i)$ are sufficiently small.

The use of these equations may be extended over multiple reaction coordinates, thus allowing the use of multidimensional umbrellas.[47,48]

Adaptive Umbrella Sampling

The use of the WHAM equations allows the facile unbiasing and combination of multiple umbrella sampling simulations. The method does not however solve the main problem of umbrella sampling, namely that of identifying the best umbrella potential. The best umbrella potential to use would be the negative of the PMF,

$$
U(\lambda) = kT \ln P_{Boltz}(\lambda), \tag{A.20}
$$

where $P_{Boltz}(\lambda)$ is the Boltzmann probability for each value of λ. This is the optimal umbrella, as it yields even sampling of the reaction coordinate.[36] Recalling that equation A.17 shows that the biased probability along λ,

$P_b(\lambda)$, is proportional to the Boltzmann probability, $P_{Boltz}(\lambda)$,

$$P_b(\lambda)\exp\big(U(\lambda)/kT\big) \propto P_{Boltz}(\lambda)$$

Multiply both sides by $\exp\big(-U(\lambda)/kT\big)$

$$P_b(\lambda) \propto P_{Boltz}(\lambda) \times \exp\big(-U(\lambda)/kT\big)$$

Using the 'best' umbrella potential from equation A.20

$$P_b(\lambda) \propto P_{Boltz}(\lambda) \times \exp\big(-kT\ln P_{Boltz}(\lambda)/kT\big) \tag{A.21}$$

$$P_b(\lambda) \propto P_{Boltz}(\lambda) \times \exp\big(-\ln P_{Boltz}(\lambda)\big)$$

$$P_b(\lambda) \propto P_{Boltz}(\lambda)/P_{Boltz}(\lambda)$$

Thus the biased sampling along λ will be even,

$$P_b(\lambda) \propto 1.$$

Adaptive Umbrella Sampling uses iterative simulations to refine an initial estimate of the umbrella potential until it is equal to the negative of the PMF.[39,40] A modification to adaptive umbrella sampling uses the WHAM equations to combine the statistics of each iteration. The combination of these two techniques is known as Adaptive Umbrella WHAM[38,37] (AdUmWHAM). The simulations are performed using the following protocol;[37,49]

1. An initial simulation is performed using a null, or zero umbrella. The system is free to sample the λ reaction coordinate. This coordinate has been divided into a series of histogram bins, i. The number of times that the system spends in each of these bins, $n_0(i)$, is recorded throughout the simulation.

2. The probability density for each bin along the reaction coordinate, $P_0(i)$, is estimated from the sampling histogram via,

$$P_0(i) = n_0(i)/N, \tag{A.22}$$

   where N is the total number of simulation steps.

3. A new umbrella potential for the next simulation, $U_1(i)$, is estimated from the estimated probability density via,

$$\begin{aligned} U_1(i) &= kT\ln P_0(i) \text{ for } P_0(i) \neq 0 \\ U_1(i) &= U_{min} \text{ for } P_0(i) = 1, \end{aligned} \tag{A.23}$$

where U_{min} is the minimum value of the umbrella from all of the occupied bins. To prevent discontinuities, the umbrella potential can be processed via a smoothing function. A suitable smoothing function may replace each value of the umbrella in each bin, $U(i)$, by,[37]

$$U(i) = \frac{1}{3}\big(-0.3U(i-2) + 1.3U(i-1) + U(i) + 1.3U(i+1) - 0.3U(i+2)\big). \tag{A.24}$$

Multiple passes of this function may be used, and a continuous umbrella may be returned via fitting to a series of functions. The number of functions should not exceed the number of bins, and typically a combination of simple polynomials, sine and cosine functions are used.[37,47,49]

4. This umbrella is then used to bias a new simulation. The statistics along the reaction coordinate are collected into a new histogram.

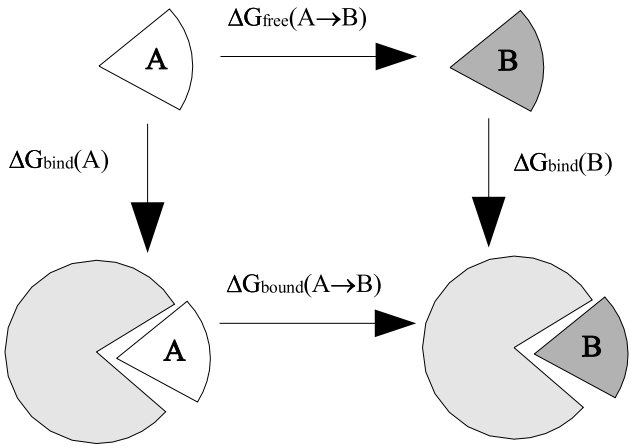
5. This histogram, together with the umbrella potential, are processed via the WHAM equations. These unbias the sampling, and combine it with all previous iterations to estimate a new probability density along the reaction coordinate, $P_1(i)$. This is used to obtain a new refinement of the umbrella potential, $U_2(i)$. This umbrella may be processed and smoothed as in step 3.

6. This new umbrella is used to bias a new simulation, which leads to a new probability histogram. This can be processed via WHAM to unbias it and combine it with all previous iterations, returning a new refinement of the umbrella potential. This sequence is then repeated until even sampling along the reaction coordinate has been achieved.

7. Once even sampling has been achieved, the umbrella equals the negative of the PMF.[49] Relative free energies along the PMF can be obtained by taking differences of points along the umbrella potential.[49]

A.3 Relative Binding Free Energies

The relative binding free energy between two ligands for the same protein can be calculated by morphing the first ligand into the second. From the free energy cycle[31] in figure A.4 it is seen that this perturbation must be performed while the ligands are bound to the protein (the bound leg), and while the ligands are free in solvent (the free leg). This is due to the fact that any host-guest binding can be viewed from the perspective of a competition between the host and solvent for the guest. Thus when we ask the question of which of a pair of ligands binds best



$$\Delta G_{bind}(B) - \Delta G_{bind}(A) = \Delta G_{bound}(A \rightarrow B) - \Delta G_{free}(A \rightarrow B) \tag{A.25}$$

Figure A.4: The free energy cycle used to calculate relative binding free energies of ligands *A* and *B* to a protein (shown in grey). The relative binding free energy of ligands *A* and *B* to a protein is equal to the perturbation free energy of *A* to *B* while bound to the protein, minus the perturbation free energy of the free ligands in solvent.

to a protein, we are really asking which ligand has the greater affinity for the protein, and the lower affinity for the solvent.

Appendix B

Monte Carlo Sampling

Normal Monte Carlo randomly generates configurations of the system, and weights each one according to its probability within the ensemble. However, these probabilities are rarely known a priori. Metropolis Monte Carlo[50] cleverly solves this by randomly generating each configuration such that it appears with its correct ensemble probability, and then weighting each configuration in the set equally. The Metropolis solution can achieve this, as it includes the Boltzmann equation. Metropolis Monte Carlo takes a configuration of the system, and calculates its energy. A random move is made, and the energy recalculated.[50] A Monte Carlo test uses this change in energy, ΔE, to evaluate the move, via,[50]

$$\exp(-\Delta E/kT) \geq rand(0,1). \tag{B.1}$$

If this test is passed, then the new configuration is accepted, otherwise the old configuration is restored, and is recounted in the average. Using this method, at the end of the MC run, the thermally important regions of the energy surface have been explored with the correct probability. This particular Monte Carlo test will sample from the canonical, NVT ensemble. Different tests exist which sample from different ensembles, e.g. isothermal-isobaric, NPT. Some of these tests are derived below.

Metropolis Monte Carlo works because it builds up a Markov chain of configurations. A Markov chain is a sequence of trials, set up such that the outcome of each trial belongs to a finite set of outcomes, and that the outcome of each trial only depends on the outcome of the trial that immediately preceded it.[51] The Markov chain can be constructed such that as the entire chain is generated, each member of the set appears in the chain with a probability that converges onto a limiting value. The aim of Metropolis Monte Carlo is to construct a Markov chain such that this limiting probability distribution is equal to the Boltzmann distribution for the ensemble for the simulation. The properties of a Markov chain mean that the final limiting probability distribution is independent of the initial guessed probability distribution. This property can be used derive the trials that are used to move

from one member of the set to the next.

$\rho^{(1)}$ represents the initial guessed probability distribution. π represents the transition matrix. This matrix operates on $\rho^{(1)}$ to produce a new estimate of the probability distribution, $\rho^{(2)}$,

$$\rho^{(2)} = \rho^{(1)}\pi. \tag{B.2}$$

As long as the conditions of the Markov chain are adhered to, then π has the property of making the probability distribution converge onto a limiting value, ρ,

$$\begin{aligned} \rho^{(3)} &= \rho^{(2)}\pi = \rho^{(1)}\pi\pi = \rho^{(1)}\pi^2 \\ \rho &= \lim_{t\to\infty} \rho^{(1)}\pi^t. \end{aligned} \tag{B.3}$$

It is clear from this equation that the limiting probability distribution must satisfy the eigenvalue equation,

$$\begin{aligned} \rho\pi &= \rho \\ \sum_m \rho_m \pi_{mn} &= \rho_n. \end{aligned} \tag{B.4}$$

It is a property of the transition matrix that its rows sum to one,

$$\sum_n \pi_{mn} = 1, \tag{B.5}$$

and that the limiting distribution, ρ, is independent of the initial starting distribution.

The aim of Metropolis Monte Carlo is to generate a transition matrix such that the limiting distribution is equal to the Boltzmann distribution, thus $\rho(i) = p_{Boltzmann}(i)$, for each point, i, in phase space. It is possible to generate the elements of the transition matrix, π_{mn}, and in so doing, generate the correct ensemble. To achieve this, we must solve the eigenvalue equation. Metropolis Monte Carlo achieves this by imposing the unnecessarily strict condition of microscopic reversibility,

$$\rho_m \pi_{mn} = \rho_n \pi_{nm}. \tag{B.6}$$

This means that the trajectory must be reversible, namely that it should be equally likely to move from state m to state n as it is to move from state n to state m. We can show that this condition satisfies the eigenvalue equation via,

$$\sum_m \rho_m \pi_{mn} = \sum_m \rho_n \pi_{nm} = \rho_n \sum_m \pi_{nm} = \rho_n. \tag{B.7}$$

The Metropolis solution solves this equation through the use of a symmetrical stochastic matrix,[52] α. This matrix shows that random moves should be attempted between states m and n, and that the probability of attempting the move from m to n should be equal to the probability of attempting to move from state n to m, i.e. $\alpha_{mn} = \alpha_{nm}$. The Metropolis solution recasts π_{mn} in terms of α, for three different cases,

$$\begin{aligned} \pi_{mn} &= \alpha_{mn} \quad \text{for all} \quad m \neq n \quad \text{and} \quad \rho_n \geq \rho_m, \\ \pi_{mn} &= \alpha_{mn}\left(\rho_n/\rho_m\right) \quad \text{for all} \quad m \neq n \quad \text{and} \quad \rho_n < \rho_m, \\ \pi_{mm} &= 1 - \sum_{n \neq m} \pi_{mn}. \end{aligned} \tag{B.8}$$

These can be shown to agree with equation B.6,

$$\begin{aligned} \rho_m \pi_{mn} &= \rho_n \pi_{nm}, \\ \text{using} \quad \rho_n &\geq \rho_m, \\ \rho_m \alpha_{mn} &= \rho_n \alpha_{nm}\left(\rho_m/\rho_n\right), \\ \rho_m \alpha_{mn} &= \alpha_{nm}\rho_m, \\ \text{since} \quad \alpha_{mn} &= \alpha_{nm} \\ \rho_m &= \rho_m. \end{aligned} \tag{B.9}$$

These equations show that the Monte Carlo simulation should be performed by attempting random moves from state m to state n with an equal probability to attempting moves from state n to state m. If $\rho_n \geq \rho_m$, then the move should be automatically accepted. If $\rho_n < \rho_m$, then the move should be accepted according to a probability of ρ_n/ρ_m. To do this, this ratio is compared to a random number between 0 and 1. If the ratio is greater than the random number, then the move is accepted. If the ratio is less than or equal to the random number, then the move is rejected, and configuration m is recounted in the average. This ratio will have different forms depending on the desired ensemble for the simulation. For example, in the NVT, canonical ensemble,

$$\begin{aligned} \rho_n/\rho_m &= \frac{\exp(-E_n/kT)/Q_{NVT}}{\exp(-E_m/kT)/Q_{NVT}} \\ &= \frac{\exp(-E_n/kT)}{\exp(-E_m/kT)} \\ &= \exp(-(E_n - E_m)/kT) \\ &= \exp(-\Delta E/kT), \end{aligned} \tag{B.10}$$

while in the NPT ensemble,

$$\begin{aligned}\rho_n/\rho_m &= \frac{\exp\left(-\frac{E_n+PV_n}{kT}+N\ln V_n\right)/Q_{NPT}}{\exp\left(-\frac{E_m+PV_m}{kT}+N\ln V_m\right)/Q_{NPT}} \\ &= \exp\left(-\frac{\Delta E+P\Delta V}{kT}+N\Delta\ln V\right).\end{aligned} \tag{B.11}$$

The Metropolis solution is only one of many solutions to the Monte Carlo equations. In preferential sampling,[53,51] an asymmetric underlying stochastic matrix is used. This allows some moves to be attempted with a higher probability than others, and that the probability of moving in one direction does not have to be equal to the probability of moving back again. For this to satisfy the condition of microscopic reversibility, the values of π_{mn} are given by,

$$\begin{aligned}&\pi_{mn} = \alpha_{mn} \text{ for all } m \neq n \text{ and } \alpha_{nm}\rho_n \geq \alpha_{mn}\rho_m, \\ &\pi_{mn} = \alpha_{mn}\left(\frac{\alpha_{nm}\rho_n}{\alpha_{mn}\rho_m}\right) \text{ for all } m \neq n \text{ and } \alpha_{nm}\rho_n < \alpha_{mn}\rho_m, \\ &\pi_{mm} = 1-\sum_{n\neq m}\pi_{mn}.\end{aligned} \tag{B.12}$$

The agreement with microscopic reversibility is shown by,

$$\begin{aligned}\pi_{mn}\rho_m &= \pi_{nm}\rho_n \\ \alpha_{mn}\rho_m &= \alpha_{nm}\left(\frac{\alpha_{mn}\rho_m}{\alpha_{nm}\rho_n}\right)\rho_n \\ \alpha_{mn}\rho_m &= \alpha_{mn}\rho_m.\end{aligned} \tag{B.13}$$

Preferential sampling may be used to increase the probability of sampling solvent molecules that are closest to the central solute. This can increase the sampling of the primary solvation shell, which is likely to have the biggest impact on any relative free energies of the solute.[51] A solvent molecule is picked with a probability, W, which decreases with distance, r, from the central solute,

$$W(r_i) = \frac{1}{r_i^v} \text{ where } v \text{ is a chosen integer parameter.} \tag{B.14}$$

In ProtoMS, v is equal to 1, and the function used is,[54]

$$W(r_i) = \frac{1}{r_i + w_{kc}}. \tag{B.15}$$

The preferential sampling constant, w_{kc} is added to the distance to help prevent the volume from continually expanding during the simulation.[54]

For each configuration, the weight of each solvent molecule is calculated, and all of the weights are normalised,

$$W'(r_i) = \frac{W(r_i)}{\sum_j W(r_j)}. \tag{B.16}$$

A solvent molecule is chosen with a probability of $W'(r_i)$. This is achieved by randomly choosing a solvent molecule, and comparing its weight against a random number between 0 and 1. If the weight is greater than the random number, then that solvent molecule is selected, otherwise a new solvent molecule is chosen and the test repeated. A Monte Carlo move is performed on the chosen solvent molecule. The weight of the solvent before the MC move, W_{old}, and weight after the move, W_{new}, are used in the modified acceptance test,

$$\frac{W_{new}}{W_{old}} \exp(-\Delta E/kT) \geq rand(0,1). \tag{B.17}$$

This test is applied in the canonical ensemble, and is derived in a similar manner to equation B.10,

$$\begin{aligned} \alpha_{mn}\left(\frac{\alpha_{nm}\rho_n}{\alpha_{mn}\rho_m}\right) &= \alpha_{nm}\frac{\rho_n}{\rho_m} \\ &= \frac{W_{new}}{W_{old}} \times \frac{\rho_n}{\rho_m} \\ &= \frac{W_{new}}{W_{old}} \exp(-\Delta E/kT). \end{aligned} \tag{B.18}$$

Index

Bibliography

[1] R. W. Zwanzig, *J. Chem. Phys.*, **22**, 1420, (1954).

[2] M. L. P. Price and W. L. Jorgensen, *J. Am. Chem. Soc.*, **122**, 9455, (2000).

[3] D. Q. McDonald and W. C. Still, *J. Am. Chem. Soc.*, **118**, 2073, (1996).

[4] D. J. Price and W. L. Jorgensen, *Bioorg. Med. Chem. Lett.*, **10**, 2067, (2000).

[5] D. J. Price and W. L. Jorgensen, *J. Comput. Aid. Mol. Des.*, **15**, 681, (2001).

[6] M. L. Lamb and W. L. Jorgensen, *J. Med. Chem.*, **41**, 3928, (1998).

[7] J. Vacek and P. A. Kollman, *J. Phys. Chem. A*, **103**, 10015, (1999).

[8] A. R. Leach, *Molecular Modelling, Principles and Applications*, Longman, Harlow, UK, (1996).

[9] W. L. Jorgensen and C. Ravimohan, *J. Chem. Phys.*, **83**, 3050, (1985).

[10] D. A. Pearlman and P. A. Kollman, *J. Chem. Phys.*, **90**, 2460, (1989).

[11] D. A. Pearlman and P. S. Charifson, *J. Med. Chem.*, **44**, 3417, (2001).

[12] B. C. Oostenbrink, J. W. Pitera, M. M. H. Lipzig, J. H. N. Meerman and W. F. Van Gunsteren, *J. Med. Chem.*, **43**, 4594, (2000).

[13] X. Barril, M. Orozco and F. J. Luque, *J. Med. Chem.*, **42**, 5110, (1999).

[14] J. T. Wescott, L. R. Fisher and S. Hanna, *J. Chem. Phys.*, **116**, 2361, (2002).

[15] J. R. Blas, M. Marquez, J. L. Sessler, F. J. Luque and M. Orozco, *J. Am. Chem. Soc.*, **124**, 12796, (2002).

[16] A. Melo and M. J. Ramos, *J. Mol. Struc-Theochem.*, **580**, 251, (2002).

[17] M. Mezei, *J. Chem. Phys.*, **86**, 7084, (1987).

[18] C. R. W. Guimaraes and R. B. Alencastro, *Int. J. Quantum Chem.*, **85**, 713, (2001).

[19] C. R. W. Guimaraes and R. B. Alencastro, *J. Med. Chem.*, **45**, 4995, (2002).

[20] C. R. W. Guimaraes and R. B. Alencastro, *J. Phys. Chem. B*, **106**, 466, (2002).

[21] S. Kamath, E. Coutinho and P. Desai, *J. Biomol. Struct. Dyn.*, **16**, 1239, (1999).

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in Fortran 77: The Art of Scientific Computing, Second edition*, Cambridge University Press, Cambridge, UK, (1996).

[23] C. J. Woods, M. A. King and J. W. Essex, *J. Phys. Chem. B*, **107**, 13703, (2003).

[24] C. J. Woods, M. A. King and J. W. Essex, *J. Phys. Chem. B*, **107**, 13711, (2003).

[25] H. Fukunishi, O. Watanabe and S. Takada, *J. Chem. Phys.*, **116**, 9058, (2002).

[26] U. H. E. Hansmann, *Chem. Phys. Lett.*, **281**, 140, (1997).

[27] T. Okabe, M. Kawata, Y. Okamoto and M. Mikami, *Chem. Phys. Lett.*, **335**, 435, (2001).

[28] C. Jarzynski, *Phys. Rev. E*, **56**, 5018, (1997).

[29] C. Jarzynski, *Phys. Rev. Lett.*, **78**, 2690, (1997).

[30] H. Hu, R. H. Yun and J. Hermans, *Mol. Simulat.*, **28**, 67, (2002).

[31] C. A. Reynolds, P. M. King and W. G. Richards, *Mol. Phys.*, **76**, 251, (1992).

[32] G. Hummer, *J. Chem. Phys.*, **114**, 7330, (2001).

[33] C. Jarzynski, *Proc. Natl. Acad. Sci.*, **98**, 3636, (2001).

[34] D. A. Hendrix and C. Jarzynski, *J. Chem. Phys.*, **114**, 5974, (2001).

[35] G. Hummer, *Mol. Simulat.*, **28**, 81, (2002).

[36] B. Roux, *Comput. Phys. Commun.*, **91**, 275, (1995).

[37] C. Bartels and M. Karplus, *J. Comput. Chem.*, **18**, 1450, (1997).

[38] S. Kumar, P. W. Payne and M. Vasquez, *J. Comput. Chem.*, **17**, 1269, (1996).

[39] R. W. W. Hooft, B. P. Vaneijck and J. Kroon, *J. Chem. Phys.*, **97**, 6690, (1992).

[40] O. Engkvist and G. Karlstrom, *Chem. Phys.*, **213**, 63, (1996).

[41] S. Kumar, D. Bouzida, R. H. Swendsen, P. A. Kollman and J. M. Rosenberg, *J. Comput. Chem.*, **13**, 1011, (1992).

[42] X. J. Kong and C. L. Brooks, *J. Chem. Phys.*, **105**, 2414, (1996).

[43] S. Banba, Z. Y. Guo and C. L. Brooks, *J. Phys. Chem. B*, **104**, 6903, (2000).

[44] Z. Guo, C. L. Brooks and X. Kong, *J. Phys. Chem. B*, **102**, 2032, (1998).

[45] G. M. Torrie and J. P. Valleau, *J. Comput. Phys.*, **23**, 187, (1977).

[46] T. N. Heinz, W. F. Van Gunsteren and P. H. Hunenberger, *J. Chem. Phys.*, **115**, 1125, (2001).

[47] C. Bartels, M. Schaefer and M. Karplus, *J. Chem. Phys.*, **111**, 8048, (1999).

[48] S. Kumar, J. M. Rosenberg, D. Bouzida, R. H. Swendsen and P. A. Kollman, *J. Comput. Chem.*, **16**, 1339, (1995).

[49] C. J. Woods, S. Camiolo, M. E. Light, S. J. Coles, M. B. Hursthouse, M. A. King, P. A. Gale and J. W. Essex, *J. Am. Chem. Soc.*, **124**, 8644, (2002).

[50] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, *J. Chem. Phys.*, **21**, 1087, (1953).

[51] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, Oxford, UK, (2001).

[52] P. W. Atkins, *Physical Chemistry, Fifth Edition*, Oxford University Press, Oxford, UK, (1995).

[53] J. C. Owicki and H. A. Scheraga, *Chem. Phys. Lett.*, **47**, 600, (1977).

[54] W. L. Jorgensen, *J. Phys. Chem.*, **87**, 5304, (1983).